L Number	Hits	Search Text	DB	Time stamp
1	25	((nic or (network adj2 (interface or adapter or controller))) same filter$3) and ((frame or packet or cell) adj classif$8)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 12:44
2	74	((nic or (network adj2 (interface or adapter or controller))) same filter$3) and ((frame or packet or cell) same classif$8)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 12:46
3	22	(((nic or (network adj2 (interface or adapter or controller))) same filter$3) and ((frame or packet or cell) same classif$8)) and @ad<19981201	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 12:46
-	2	bandwidth adj broker$3	USPAT; EPO; DERWENT; USOCR	2001/09/22 18:52
-	22	bandwidth same broker$3	USPAT; EPO; DERWENT; USOCR	2001/09/22 18:56
-	11936	service near20 level$1	USPAT; EPO; DERWENT; USOCR	2001/09/22 19:05
-	3561	tos or qos	USPAT; EPO; DERWENT; USOCR	2001/09/22 19:07
-	342	(service near20 level$1) and (tos or qos)	USPAT; EPO; DERWENT; USOCR	2001/09/22 19:06
-	1	(bandwidth same broker$3) and ((service near20 level$1) and (tos or qos))	USPAT; EPO; DERWENT; USOCR	2001/09/22 19:06
-	2441	tos	USPAT; EPO; DERWENT; USOCR	2001/09/22 19:07
-	4	tos same (service near20 level$1)	USPAT; EPO; DERWENT; USOCR	2001/09/22 19:09
-	304	(service near20 level$1) same filter$3	USPAT; EPO; DERWENT; USOCR	2001/09/22 19:10
-	0	((service near20 level$1) same filter$3) and tos	USPAT; EPO; DERWENT; USOCR	2001/09/22 19:10
-	261	bandwidth$1 and broker$3	USPAT; EPO; DERWENT; USOCR	2001/09/22 19:10

-	3	((service near20 level$1) same filter$3) and (bandwidth$1 and broker$3)	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/22 19:11

-	4909	service near3 level$1	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/22 19:11

-	9939	admission same control$4	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/22 19:12

-	19	(service near3 level$1) same (admission same control$4)	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/22 19:21

-	9448	ingress and egress	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/22 19:21

-	3585	bandwidth adj (allocat$4 or broker$5 or control$4 or request$3)	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/22 19:22

-	57	(ingress and egress) and (bandwidth adj (allocat$4 or broker$5 or control$4 or request$3))	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/22 19:22

-	13	(service near3 level$1) and ((ingress and egress) and (bandwidth adj (allocat$4 or broker$5 or control$4 or request$3)))	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/22 19:26

-	168	(admission$1 or access) adj profil$3	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/22 19:27

-	1190	((709/226) or (709/225) or (709/229)).CCLS.	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/22 19:27

-	7	((admission$1 or access) adj profil$3) and ((("709/226") or ("709/225") or ("709/229")).CCLS.)	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/22 19:27

-	59	policy adj server$1	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/25 12:04

-	675	713/15$.ccls.	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/25 12:04

-	6	(policy adj server$1) and 713/15$.ccls.	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/25 12:07

-	1	admission$1 adj profile$1	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/25 12:29

-	4276	dynamic$5 near5 filter$3	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/25 12:09

-	2	(dynamic$5 near5 filter$3) and (policy adj server$1)	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/25 12:29

-	306	filter$3 adj criteria$1	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/25 12:30

-	39	admission$1 adj10 profile$1	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/25 12:29

-	0	(filter$3 adj criteria$1) and (admission$1 adj10 profile$1)	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/25 12:29

-	17	(dynamic$5 near5 filter$3) and (filter$3 adj criteria$1)	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/25 12:35

-	47	ingress near4 profil$3	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/25 12:35

-	0	(admission$1 adj10 profile$1) and (ingress near4 profil$3)	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/25 12:35

-	0	(ingress near4 profil$3) and 713/15$.ccls.	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/25 12:35

-	3	(ingress near4 profil$3) and filter$3	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/25 12:38

-	102511	remov$3 near10 filter$1	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/25 12:40

-	84571	remov$3 near5 filter$1	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/25 12:42

-	0	(admission$1 adj10 profile$1) and (remov$3 near5 filter$1)	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/25 12:42

-	2	(policy adj server$1) and (remov$3 near5 filter$1)	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/25 12:43

-	4	713/15$.ccls. and (remov$3 near5 filter$1)	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/25 12:43

-	867	filter$3 near5 (delet$3 or expir$6)	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/26 10:48

-	8494	713/$.ccls.	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/26 10:48

-	0	(dynamically with (creat$3 or establish$3 or remov$3 or delet$3) with filter$1) and 713/$.ccls.	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/26 10:48

-	20	(filter$3 near5 (delet$3 or expir$6)) and 713/$.ccls.	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/26 12:24

-	89	dynamically with (creat$3 or establish$3 or remov$3 or delet$3) with filter$1	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/26 10:53

-	9	dynamically near3 (creat$3 or establish$3 or remov$3 or delet$3) near3 filter$1	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/26 11:45

-	2	("6148336").PN.	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/26 11:45

-	1	(("6148336").PN.) and (remov$3 or delet$3 or expir$7)	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/26 12:05

-	1	((("6148336").PN.) and (remov$3 or delet$3 or expir$7)) and packet$1	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/26 12:06

-	1	((("6148336").PN.) and (remov$3 or delet$3 or expir$7)) and (destination$1 or source$1)	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/26 12:09

-	0	(("6148336").PN.) and (admission adj profile$1)	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/26 12:10

-	0	(("6148336").PN.) and (admission near4 profile$1)	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/26 12:10

-	1	(("6148336").PN.) and polic$3	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/26 12:25

-	1	(("6148336").PN.) and (policy or policies)	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/26 12:25

-	26	(("5554322") or ("5884033") or ("5953338") or ("5968176") or ("6055571") or ("6130924") or ("6148336") or ("6167451") or ("6167445") or ("6178505") or ("6199113") or ("6256741") or ("6262974") or ("6295527")).PN.	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/29 18:39

-	3	("5544322").PN.	USPAT;
			EPO;
			DERWENT;
			USOCR
				2001/09/29 18:39

-	4260	relational adj database	USPAT;
			EPO;
			DERWENT;
			USOCR
				2002/04/11 13:05

-	11775	707/$.ccls.	USPAT;
			EPO;
			DERWENT;
			USOCR
				2002/04/11 13:05

-	1757	(relational adj database) and 707/$.ccls.	USPAT	2002/04/11 13:06
-	296	((relational adj database) and 707/$.ccls.) and api	USPAT	2002/04/11 13:06
-	1778	trigger$3 near5 filter$1	USPAT	2002/04/18 11:26
-	15	(trigger$3 near5 filter$1) and 713/$.ccls.	USPAT	2002/04/18 11:26
-	1	("6178505").PN.	USPAT	2002/04/22 08:29
-	1	(("6178505").PN.) and expir$7	USPAT	2002/04/22 08:29
-	163	tos same filter$3	USPAT	2002/11/06 15:43
-	6	(tos same filter$3) same (type adj2 service$1)	USPAT	2002/11/06 15:45
-	4611	classif$8 same filter$3	USPAT	2002/11/06 15:45
-	5	(classif$8 same filter$3) same tos	USPAT	2002/11/06 15:46
-	14	(classif$8 same filter$3) and tos	USPAT	2002/11/06 15:46

-	52	packet adj classifier$1	USPAT;
			EPO;
			DERWENT;
			USOCR
				2002/11/14 16:05

-	26	(packet adj classifier$1) and filter$3	USPAT;
			EPO;
			DERWENT;
			USOCR
				2002/11/14 16:05

-	21	(packet adj classifier$1) and (qos or tos)	USPAT;
			EPO;
			DERWENT;
			USOCR
				2002/11/14 16:05

-	15	((packet adj classifier$1) and filter$3) and ((packet adj classifier$1) and (qos or tos))	USPAT;
			EPO;
			DERWENT;
			USOCR
				2002/11/14 16:14

-	4	("5636371" | "5729685" | "5802286" | "5826014").PN.	USPAT	2002/11/14 16:13

-	68	witting.inv.	USPAT;
			EPO;
			DERWENT;
			USOCR
				2002/11/14 16:14

-	2	witting.inv. and filter$3	USPAT;
			EPO;
			DERWENT;
			USOCR
				2002/11/14 16:15

-	1	witting.inv. and packet$1	USPAT;
			EPO;
			DERWENT;
			USOCR
				2002/11/14 16:15

-	253	barzilai	USPAT;
			EPO;
			DERWENT;
			USOCR
				2002/11/14 16:15

-	17	barzilai.inv.	USPAT;
			EPO;
			DERWENT;
			USOCR
				2002/11/14 16:16

-	77	barzilai and packet$1	USPAT; EPO; DERWENT; USOCR	2002/11/14 16:16
-	9	(barzilai and packet$1) and classif$9	USPAT; EPO; DERWENT; USOCR	2002/11/14 16:18
-	16	"5040176"	USPAT; EPO; DERWENT; USOCR	2002/11/14 16:18
-	2	5040176.pn.	USPAT; EPO; DERWENT; USOCR	2002/11/14 16:20
-	5	tsipora and barzilai.inv.	USPAT; EPO; DERWENT; USOCR	2002/11/14 16:20
-	0	(tsipora and barzilai.inv.) and filter$3	USPAT; EPO; DERWENT; USOCR	2002/11/14 16:21
-	0	(tsipora and barzilai.inv.) and qos	USPAT; EPO; DERWENT; USOCR	2002/11/14 16:21
-	1	(tsipora and barzilai.inv.) and classification	USPAT; EPO; DERWENT; USOCR	2002/11/14 16:24
-	13	wittig and hartmut	USPAT; EPO; DERWENT; USOCR	2002/11/14 16:29
-	3	(("6084879") or ("6335935") or ("6157955")).PN.	USPAT	2002/11/14 16:31
-	522	packet adj classif$8	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 11:12
-	287	(nic or (network adj (interface or adapter))) near5 filter$3	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 11:13
-	60477	370/$.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 11:09
-	74	370/$.ccls. and ((nic or (network adj (interface or adapter))) near5 filter$3)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 11:10
-	8	(370/$.ccls. and ((nic or (network adj (interface or adapter))) near5 filter$3)) and classif$8	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 11:10

-	13	(packet adj classif$8) and ((nic or (network adj (interface or adapter))) near5 filter$3)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 11:10
-	1303	(nic or (network adj (interface or adapter or controller))) same filter$3	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 11:13
-	1511	(frame or packet or cell) adj classif$8	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 11:13
-	311	(nic or (network adj2 (interface or adapter))) near5 filter$3	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 11:13
-	1466	(nic or (network adj2 (interface or adapter or controller))) same filter$3	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 11:14
-	15	((nic or (network adj2 (interface or adapter))) near5 filter$3) and ((frame or packet or cell) adj classif$8)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 11:16
-	7	("5774660" | "5918021" | "6151297" | "6160544" | "6243360" | "6253334" | "6314525").PN.	USPAT	2003/04/30 11:15
-	4	(("5774660" | "5918021" | "6151297" | "6160544" | "6243360" | "6253334" | "6314525").PN.) and filter$3	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 11:18
-	25	((nic or (network adj2 (interface or adapter or controller))) same filter$3) and ((frame or packet or cell) adj classif$8)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 12:43

US006167445A

United States Patent [19]

Gai et al.

[11] **Patent Number: 6,167,445**

[45] **Date of Patent: Dec. 26, 2000**

[54] **METHOD AND APPARATUS FOR DEFINING AND IMPLEMENTING HIGH-LEVEL QUALITY OF SERVICE POLICIES IN COMPUTER NETWORKS**

[75] Inventors: **Silvano Gai**, Vigliano d'Asti, Italy; **Keith McCloghrie**, San Jose, Calif.

[73] Assignee: **Cisco Technology, Inc.**, San Jose, Calif.

[21] Appl. No.: **09/179,036**

[22] Filed: **Oct. 26, 1998**

[51] **Int. Cl.**[7] **G06F 15/177**

[52] **U.S. Cl.** **709/223**; 709/220

[58] **Field of Search** 713/200, 201; 709/220, 221, 222, 223, 224

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,769,810	9/1988	Eckberg, Jr. et al.	370/60
4,769,811	9/1988	Eckberg, Jr. et al.	370/60
5,224,099	6/1993	Corbalis et al.	370/94.2
5,263,157	11/1993	Janis	395/600
5,473,599	12/1995	Li et al.	370/16
5,606,668	2/1997	Shwed	713/201
5,666,353	9/1997	Klausmeier et al.	370/230
5,751,967	5/1998	Raab et al.	709/228
5,819,042	10/1998	Hansen	709/222
5,827,928	2/1999	Lewis et al.	709/222
5,832,503	11/1998	Malik et al.	709/222 X
5,889,953	3/1999	Thebaut et al.	709/221
5,987,513	11/1999	Prithviraj et al.	709/223
6,041,347	3/2000	Harsham et al.	709/220
6,047,322	4/2000	Vaid et al.	709/224

OTHER PUBLICATIONS

Ortiz, Jr., S., "Active Networks: The Programmable Pipeline", *Computer* pp. 19–21 Aug. 1998.

IEEE P802.1D Standard (draft 15) "Local and Metropolitan Area Networks", pp. 1, 50–56 and 378–381 (Nov. 1997).

"An Emerging Trend in the Internet Services Market", Hewlett-Packard Corp. (date unknown).

Wroclawaski, J., "The Use of RSVP with IETF Intergrated Services", IETF Network Working Group (Sep. 1997).

Bernet, Y. et al., "Framework for Use of RSVP with Diff-serv Networks", IETF (Nov. 1998).

Bernet, Y. et al., "Requirements of Diff-serv Boundary Routers", IETF Differentiated Services (Nov. 1998).

Yadav, S. et al., "Identity Representation for RSVP", IETF (Jan. 1999).

Heinanen, J. et al., "Assured Forwarding PHB Group", IETF (Sep. 1998).

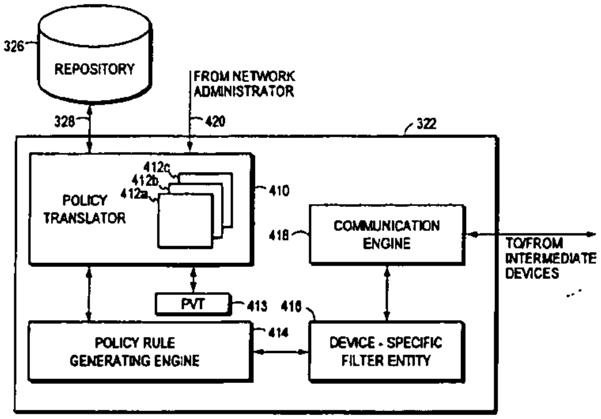
(List continued on next page.)

Primary Examiner—Zarni Maung
Assistant Examiner—Patrice Winder
Attorney, Agent, or Firm—Cesari and McKenna, LLP

[57] **ABSTRACT**

A computer network having multiple, dissimilar network devices includes a system for implementing high-level, network policies. The high-level policies, which are generally device-independent, are translated by one or more policy servers into a set of rules that can be put into effect by specific network devices. Preferably, a network administrator selects an overall traffic template for a given domain and may assign various applications and/or users to the corresponding traffic types of the template. Location-specific policies may also be established by the network administrator. The policy server translates the high-level policies inherent in the selected traffic template and location-specific policies into a set of rules, which may include one or more access control lists, and may combine several related rules into a single transaction. Intermediate network devices, which may have one or more roles assigned to their interfaces, are configured to request traffic management information from the policy server which replies with a particular set of transactions and rules. The rules, which may correspond to the particular roles assigned to the interfaces, are then utilized by the intermediate devices to configure their particular services and traffic management mechanisms. Other rules are utilized by the intermediate devices to classify packets with a particular priority and/or service value and to treat classified packets in a particular manner so as to realize the selected high-level policies within the domain.

18 Claims, 9 Drawing Sheets


326
REPOSITORY
FROM NETWORK ADMINISTRATOR
328
420
322
412c
412b
412a
POLICY TRANSLATOR
410
418
COMMUNICATION ENGINE
TO/FROM INTERMEDIATE DEVICES
PVT
413
416
414
POLICY RULE GENERATING ENGINE
DEVICE - SPECIFIC FILTER ENTITY

OTHER PUBLICATIONS

Jacobson, V. et al., "An Expedite Forwarding PHB", IETF Differentiated Services Working Group (Aug. 1998).
Nichols, K. et al., "Definition of the Differentiated Services Field (DS Field) in the IPv4 and Ipv6 Hreaders", IETF Differentiated Services Working Group (Aug. 1998).
Blake, S. et al., "An Architecture for Differentiated Services", IETF Differented Services Working Group (Aug. 1998).
Bernet, Y. et al., "A Framework for End-to-End QoS Combining RSVP/Interserv and Differentiated Services", IETF (Mar. 1998).
Yavatkar, R. et al., "A Framework for Policy-based Admission Control", IETF (Nov. 1997).
Boyle, J. et al., "The COPS (Common Open Policy Service) Protocol", IETF (Aug. 1998).
Reichmeyer, F. et al., "COPS Usage for Differentiated Services", IETF Network Working Group (Aug. 1998).
"Cisco IOS® Software Quality of Service Solutions", Cisco Systems, Inc. (Jul. 1998).
"Queuing, Traffic Shaping, and Filtering", Cisco Systems, Inc. (Sep, 1996).
"Policy-Based Routing", Cisco Systems, Inc. (Sep. 1996).
"Network Node Registry Overwiew" (Jan. 29, 1998).
"Network Node Registry User's Guide" (Apr. 1997).
"Network Node Registy—Access Control Lists" (Apr. 1997).
"Quality of Service Policy Propagation via Border Gateway Protocol", Cisco Systems, Inc. (Feb. 1998).
"Distributed Weighted Random Early Detection", Cisco Systems, Inc., pp. 1-6 (Feb. 1998).
"Distributed Weighted Fair Queuing", Cisco Systems, Inc. (Mar. 1998).
"Action Request Systems®", Remedy Corporation (1998).
"3Com's Framework for Delivering Policy-Powered Networks", 3Com Corporation (Jun. 1998).

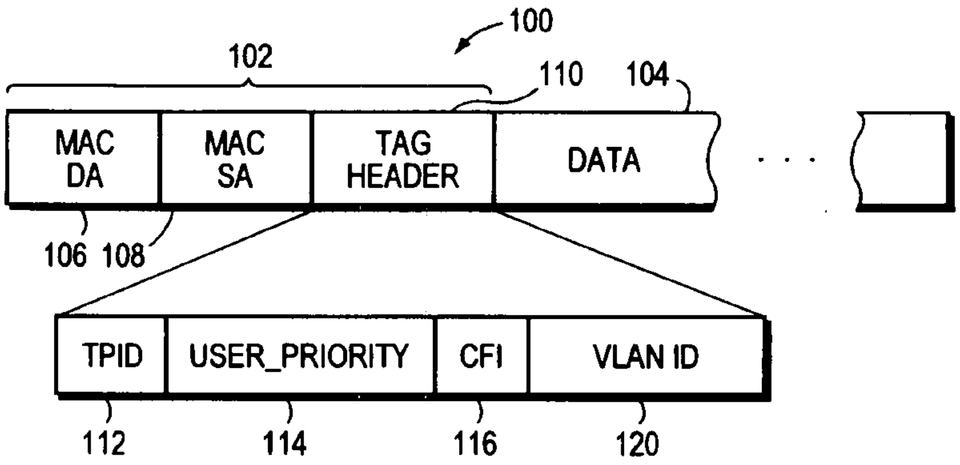


FIG. 1 (PRIOR ART)

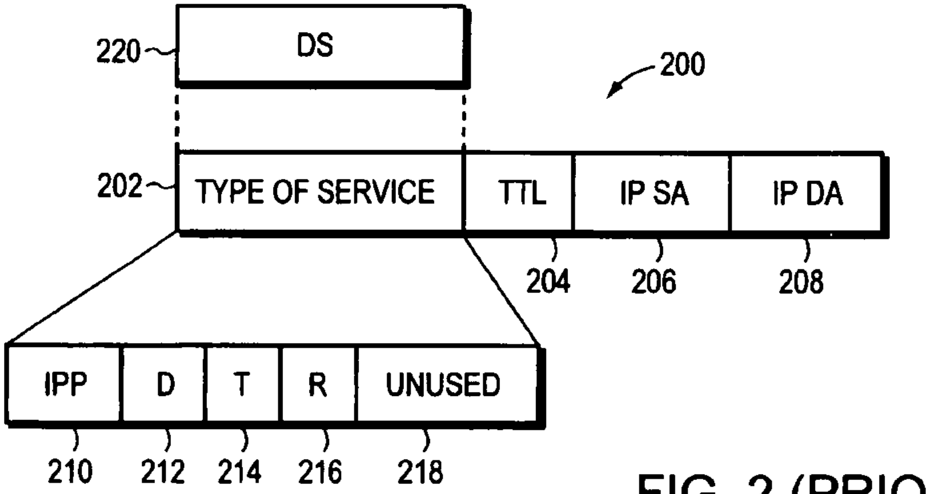


FIG. 2 (PRIOR ART)

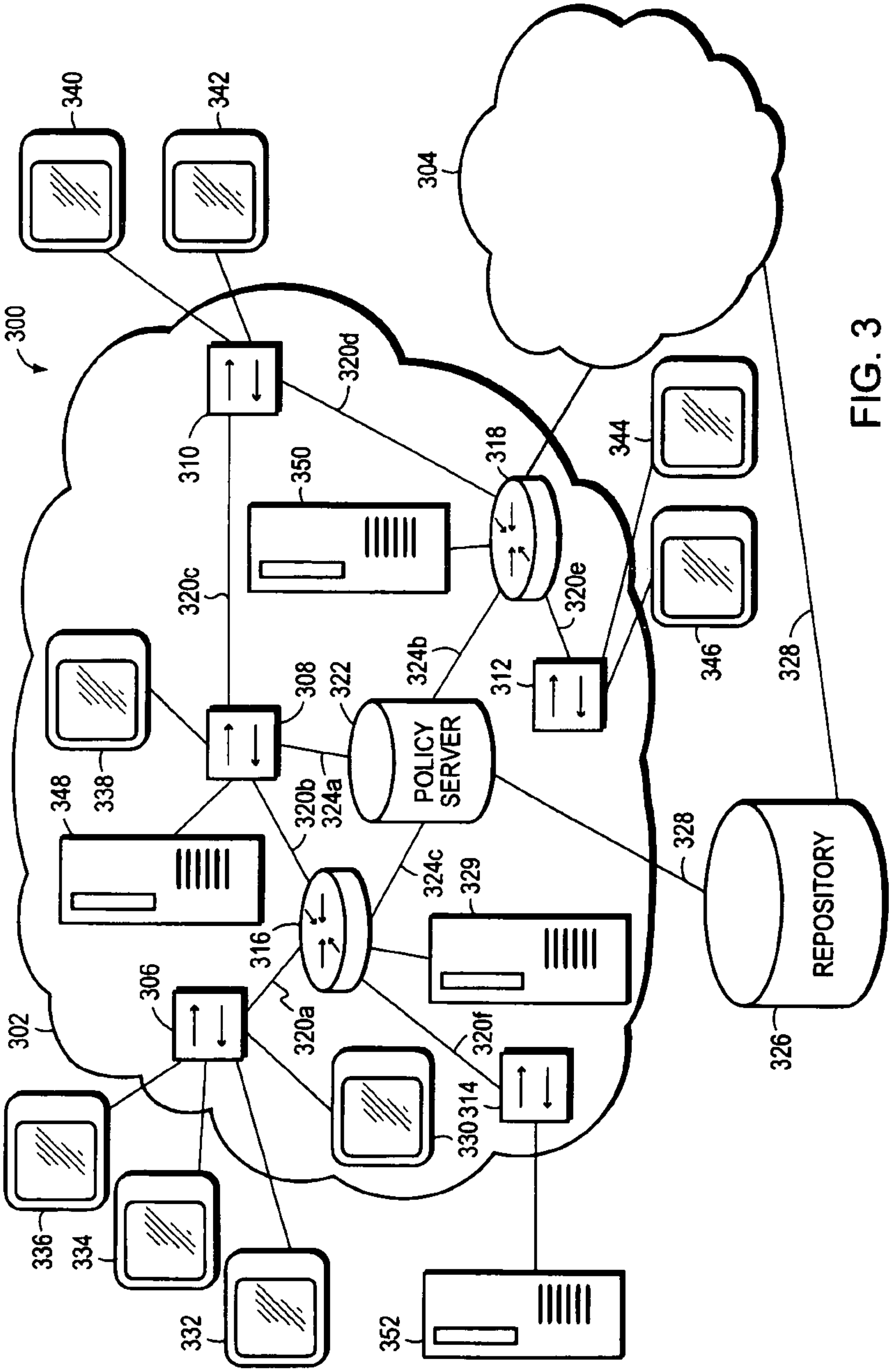

300
302
304
306
308
310
312
314
316
318
320a
320b
320c
320d
320e
320f
322
POLICY SERVER
324a
324b
324c
326
REPOSITORY
328
328
329
330
332
334
336
338
340
342
344
346
348
350
352

FIG. 3

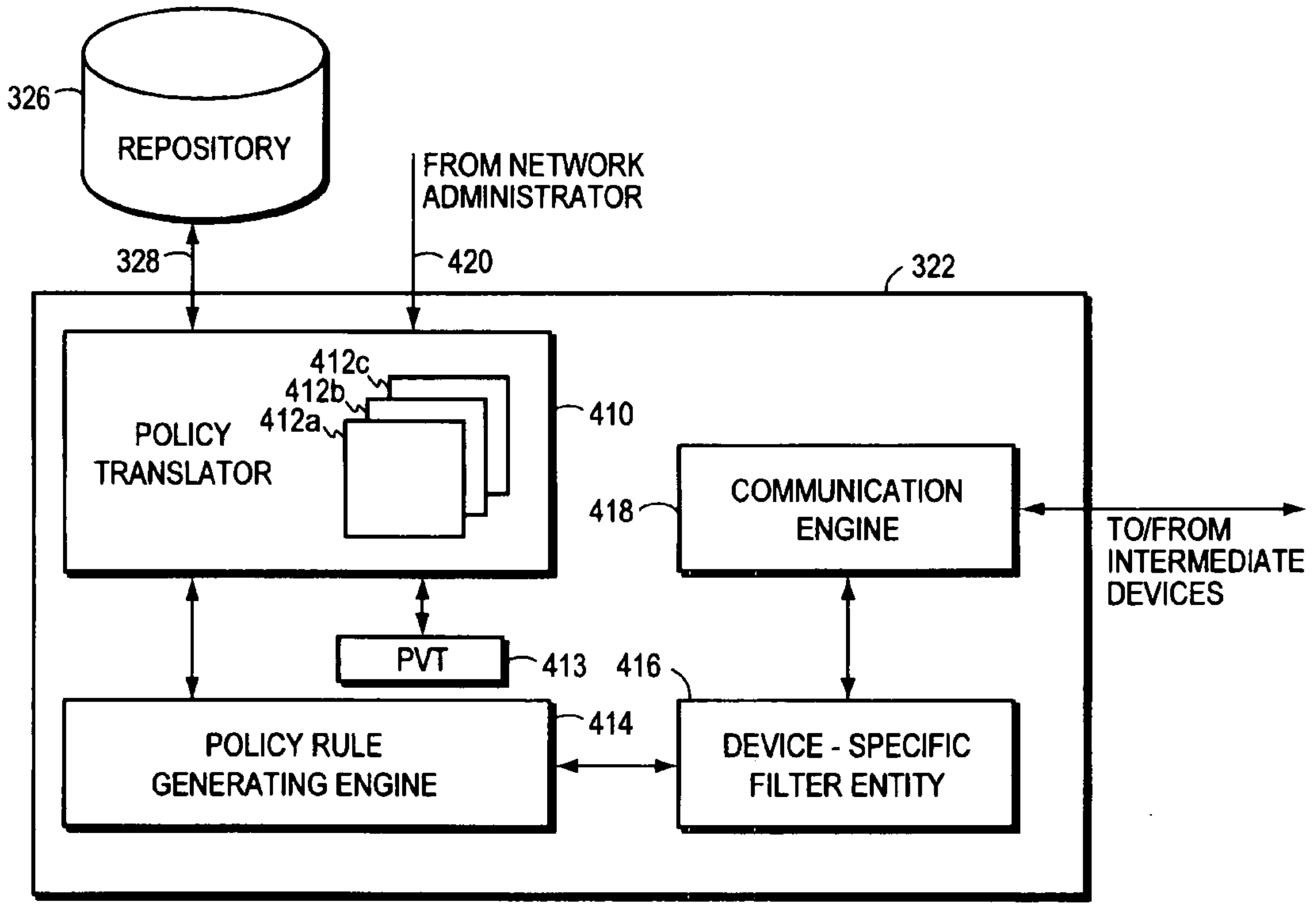

326
REPOSITORY
328
FROM NETWORK ADMINISTRATOR
420
322
POLICY TRANSLATOR
412c
412b
412a
410
418
COMMUNICATION ENGINE
TO/FROM INTERMEDIATE DEVICES
PVT
413
416
414
POLICY RULE GENERATING ENGINE
DEVICE - SPECIFIC FILTER ENTITY

FIG. 4

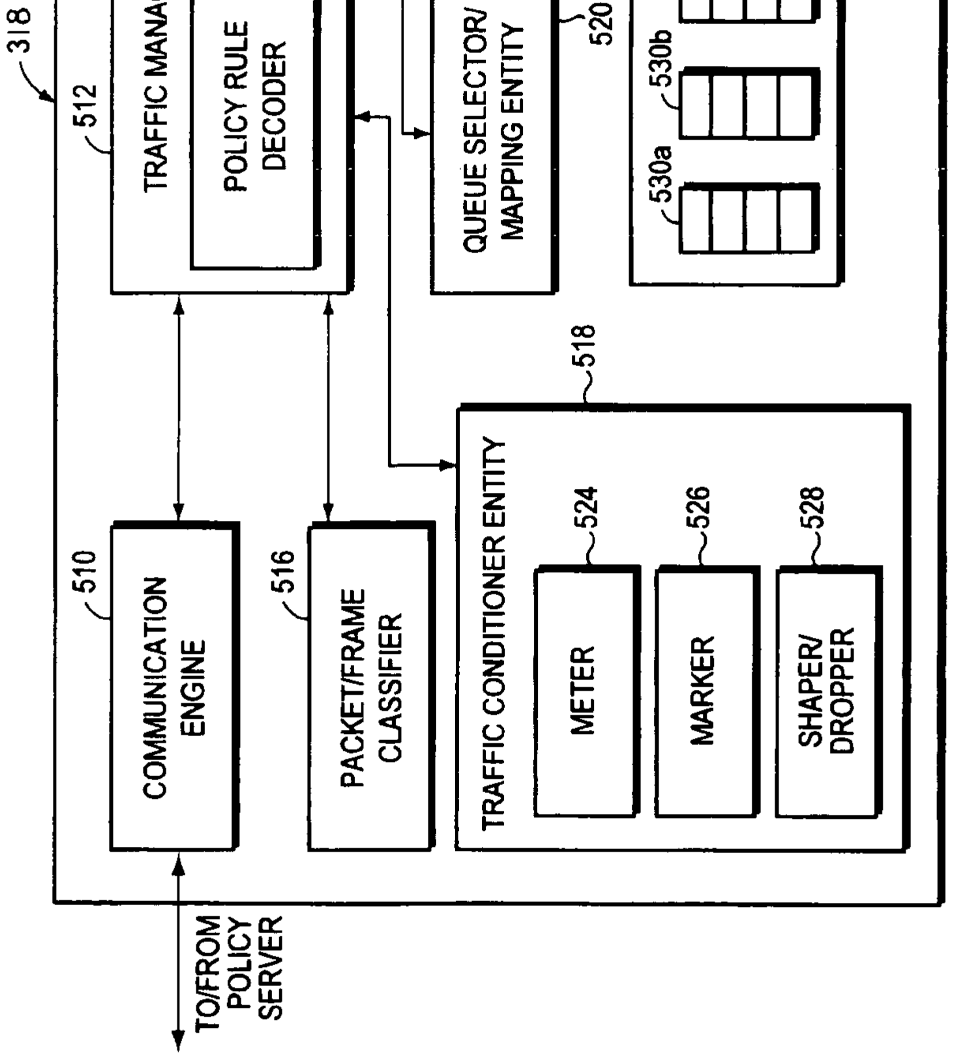
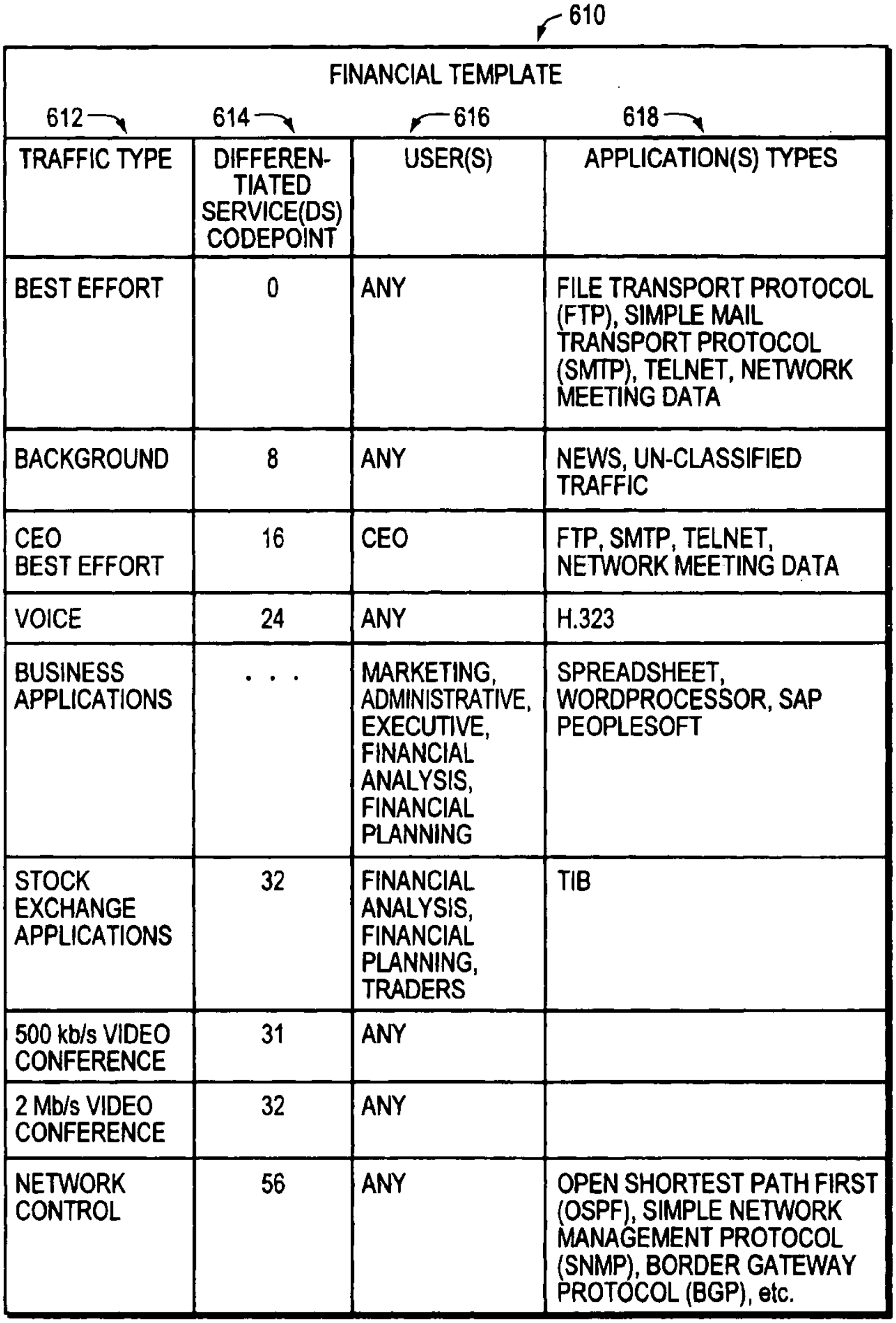

318
TRAFFIC MANAGEMENT CONTROLLER
512
POLICY RULE DECODER
514
SCHEDULER
522
QUEUE SELECTOR/ MAPPING ENTITY
520
532
530a
530b
530c
530d
530e
COMMUNICATION ENGINE
510
PACKET/FRAME CLASIFIER
516
TRAFFIC CONDITIONER ENTITY
518
METER
524
MARKER
526
SHAPER/ DROPPER
528
TO/FROM POLICY SERVER

FIG. 5


610
612
614
616
618

FINANCIAL TEMPLATE

TRAFFIC TYPE	DIFFERENTIATED SERVICE(DS) CODEPOINT	USER(S)	APPLICATION(S) TYPES
BEST EFFORT	0	ANY	FILE TRANSPORT PROTOCOL (FTP), SIMPLE MAIL TRANSPORT PROTOCOL (SMTP), TELNET, NETWORK MEETING DATA
BACKGROUND	8	ANY	NEWS, UN-CLASSIFIED TRAFFIC
CEO BEST EFFORT	16	CEO	FTP, SMTP, TELNET, NETWORK MEETING DATA
VOICE	24	ANY	H.323
BUSINESS APPLICATIONS	. . .	MARKETING, ADMINISTRATIVE, EXECUTIVE, FINANCIAL ANALYSIS, FINANCIAL PLANNING	SPREADSHEET, WORDPROCESSOR, SAP PEOPLESOFT
STOCK EXCHANGE APPLICATIONS	32	FINANCIAL ANALYSIS, FINANCIAL PLANNING, TRADERS	TIB
500 kb/s VIDEO CONFERENCE	31	ANY	
2 Mb/s VIDEO CONFERENCE	32	ANY	
NETWORK CONTROL	56	ANY	OPEN SHORTEST PATH FIRST (OSPF), SIMPLE NETWORK MANAGEMENT PROTOCOL (SNMP), BORDER GATEWAY PROTOCOL (BGP), etc.

FIG. 6

710

712 714 716 718

USER(S)	NAME	IP ADDRESS	IP MASK
CEO	JOHN DOE	[??????]	x.y.z.1
ANY	N/A		x.y.z..
MARKETING	N/A		x.y.z.2
ADMINISTRATIVE	N/A		x.y.z.3
EXECUTIVE	N/A		x.y.z.1
FINANCIAL PLANNING	N/A		x.y.z.4
FINANCIAL ANALYSTS	N/A		x.y.z.5
TRADERS	N/A		x.y.z.6

FIG. 7A

730

732 734

APPLICATION PROGRAM	NETWORK PROTOCOL/PORT NUMBER
FTP-DATA	TCP/20
FTP	TCP/21
TELNET	TCP/23
SMTP	TCP/25
TIME SERVER APPLICATION	UDP/37
NAME SERVER APPLICATION	UDP/42
H323, VOICE	@h245.voice.inspect

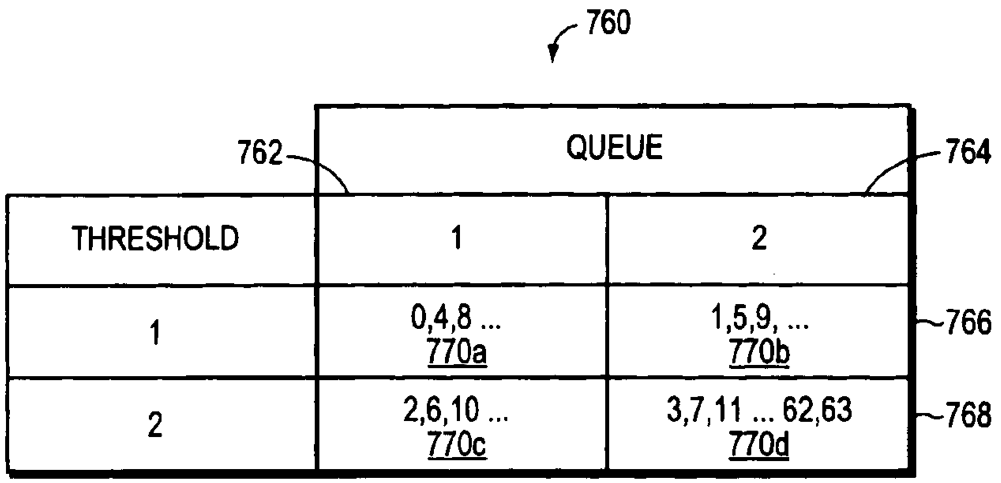
FIG. 7B

740

742 744 746 748

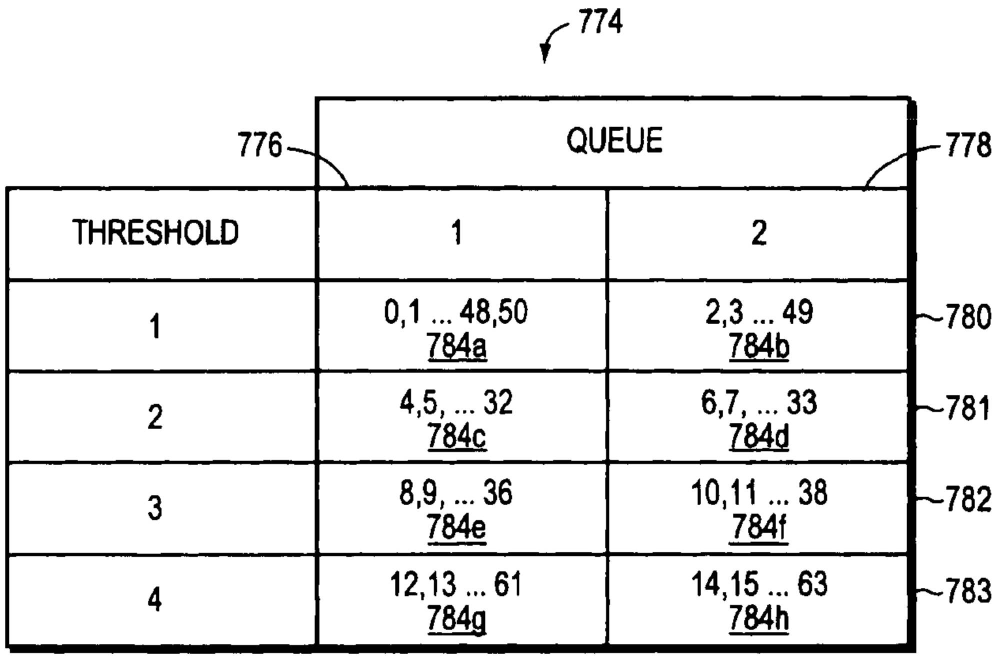
DS CODEPOINT	DS MARK DOWN VALUE	USER PRIORITY VALUE	IPP VALUE
0	0	0	0
1	0	1	1
2	1	2	2
3	1	3	3
4	1	3	3
5	1	3	3
6	1	3	3
7	1	3	3
8	1	3	3
9	1	3	3
10	1	3	3
. . .	. . .	. . .	. . .
20			
. . .	. . .	. . .	. . .
30			
. . .	. . .	. . .	. . .
40			
. . .	. . .	. . .	. . .
50			
. . .	. . .	. . .	. . .
60	14	6	6
61	14	6	6
62	28	7	7
63	28	7	7

FIG. 7C


760
762
764
766
768

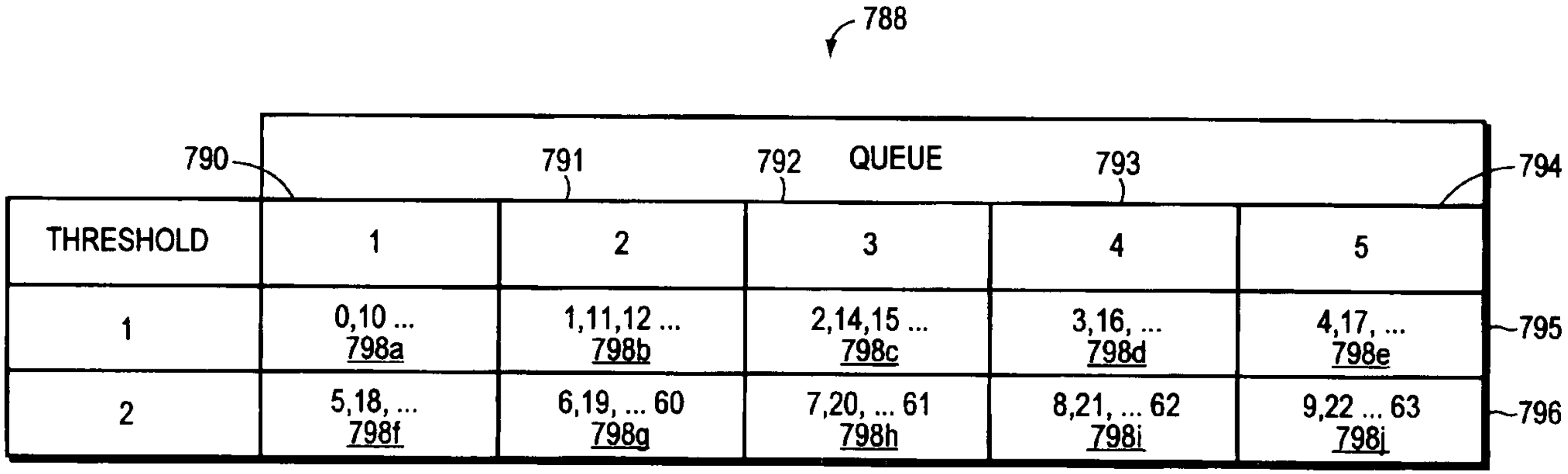
	QUEUE	
THRESHOLD	1	2
1	0,4,8 ... 770a	1,5,9, ... 770b
2	2,6,10 ... 770c	3,7,11 ... 62,63 770d

FIG. 7D


774
776
778
780
781
782
783

	QUEUE	
THRESHOLD	1	2
1	0,1 ... 48,50 784a	2,3 ... 49 784b
2	4,5, ... 32 784c	6,7, ... 33 784d
3	8,9, ... 36 784e	10,11 ... 38 784f
4	12,13 ... 61 784g	14,15 ... 63 784h

FIG. 7E


788
790
791
792
793
794
795
796

	QUEUE				
THRESHOLD	1	2	3	4	5
1	0,10 ... 798a	1,11,12 ... 798b	2,14,15 ... 798c	3,16, ... 798d	4,17, ... 798e
2	5,18, ... 798f	6,19, ... 60 798g	7,20, ... 61 798h	8,21, ... 62 798i	9,22 ... 63 798j

FIG. 7F

METHOD AND APPARATUS FOR DEFINING AND IMPLEMENTING HIGH-LEVEL QUALITY OF SERVICE POLICIES IN COMPUTER NETWORKS

FIELD OF THE INVENTION

The present invention relates generally to computer networks, and more specifically, to a method and apparatus for applying high-level, quality of service policies at dissimilar computer network devices.

BACKGROUND OF THE INVENTION

A computer network typically comprises a plurality of interconnected entities that transmit (i.e., "source") or receive (i.e., "sink") data frames. A common type of computer network is a local area network ("LAN") which typically refers to a privately owned network within a single building or campus. LANs employ a data communication protocol (LAN standard), such as Ethernet, FDDI or token ring, that defines the functions performed by the data link and physical layers of a communications architecture (i.e., a protocol stack), such as the Open Systems Interconnection (OSI) Reference Model. In many instances, multiple LANs may be interconnected by point-to-point links, microwave transceivers, satellite hook-ups, etc. to form a wide area network ("WAN"), metropolitan area network ("MAN") or intranet. These LANs and/or WANs, moreover, may be coupled through one or more gateways to the Internet.

One or more intermediate devices are often used to couple LANs together and allow the corresponding entities to exchange information. For example, a bridge may be used to provide a "bridging" function between two or more LANs. Alternatively, a switch may be utilized to provide a "switching" function for transferring information, such as data frames, among entities of a computer network. Typically, the switch is a computer having a plurality of ports that couple the switch to several LANs and to other switches. The switching function includes receiving data frames at a source port and transferring them to at least one destination port for receipt by another entity.

Switches may operate at various levels of the communication protocol stack. For example, a switch may operate at layer 2 which, in the OSI Reference Model, is called the data link layer and includes the Logical Link Control (LLC) and Media Access Control (MAC) sub-layers. Data frames at the data link layer typically include a header containing the MAC address of the entity sourcing the message, referred to as the source address, and the MAC address of the entity to whom the message is being sent, referred to as the destination address. To perform the switching function, layer 2 switches examine the MAC destination address of each data frame received on a source port. The frame is then switched onto the destination port(s) associated with that MAC destination address.

Other devices, commonly referred to as routers, may operate at higher communication layers, such as layer 3 of the OSI Reference Model, which in TCP/IP networks corresponds to the Internet Protocol (IP) layer. Data frames at the IP layer also include a header which contains an IP source address and an IP destination address. Routers or layer 3 switches may re-assemble or convert received data frames from one LAN standard (e.g., Ethernet) to another (e.g. token ring). Thus, layer 3 devices are often used to interconnect dissimilar subnetworks.

User Priority

FIG. 1 is a block diagram of a data link (e.g., Ethernet) frame 100 which includes a MAC header 102 and a data field 104. MAC header 102 includes a MAC destination address (MAC DA) field 106 and a MAC source address (MAC SA) field 108. Recently, a proposal was made to insert a new field after the MAC SA field 108. More specifically, the Institute of Electrical and Electronics Engineers (IEEE) is working on a standard, the IEEE 802.1Q draft standard, for adding information to MAC headers. In particular, the 802.1Q standard defines a tag header 110 which is inserted immediately following the MAC DA and MAC SA fields 106, 108.

The tag header 110 comprises a plurality of sub-fields, including a Tag Protocol Identifier (TPID) field 112, a user_priority field 114, a Canonical Format Indicator (CFI) field 116 and a Virtual Local Area Network Identifier (VLAN ID) field 120. The user_priority field 114 permits network devices to select a desired priority of data link frames. In particular, in an IEEE appendix, referred to as the 802.1p standard, the IEEE has defined eight possible values of user priority (0–7), each of which is associated with a specific traffic type. The proposed user priority values and corresponding traffic types specified in the 802.1p standard are as follows.

User Priority Value	Traffic Type	Description
1	Background	bulk transfers
2	Spare/Reserved	n/a
0	Best Effort	current LAN traffic
3	Excellent Effort	best effort type of services (e.g., for an organization's most important customers)
4	Controlled Load	important business applications
5	Video (<100 milliseconds latency and jitter)	minimum jitter
6	Voice (<10 milliseconds latency and jitter)	one-way transmission through the LAN
7	Network Control	characterized by a "must get there" requirement to maintain and support the network infrastructure

An intermediate device may provide a plurality of transmission priority queues per port and, pursuant to the 802.1p standard, may assign frames to different queues of a destination port on the basis of the frame's user priority value. For example, frames with a user priority of "0" are placed in the "0" level priority queue (e.g., non-expedited traffic), whereas frames with a user priority of "3" are placed in the level "3" priority queue. Furthermore, frames stored in a higher level queue (e.g., level 3/excellent effort) are preferably forwarded before frames stored in a lower level queue (e.g., level 1/background). This is commonly referred to as Priority Queuing. Thus, by setting the contents of the user_priority field 114 to a particular value, a device may affect the speed with which the corresponding frames traverse the network.

If a particular intermediate device has less than eight priority queues per port, several of the IEEE traffic types may be combined. For example, if only three queues are present, then queue 1 may accommodate best effort, excellent effort and background traffic types, queue 2 may accommodate controlled load and video traffic types and queue 3 may accommodate voice and network control traffic types. The IEEE 802.1p standard also recognizes that intermediate devices may regenerate the user priority value of a received frame. That is, an intermediate device may forward the frame with a different user priority value (still within the range of 0–7) than the one it had when the frame was received. Nevertheless, the standard recommends leaving the user priority value un-changed.

Type of Service

FIG. 2 is a block diagram of a portion of an Internet Protocol Version 4 (IPv4) compliant IP header 200. The IP header 200 is also made up of a plurality of fields, including a type_of_service (ToS) field 202, a time to live (TTL) field 204, an IP source address (IP SA) field 206 and an IP destination address (IP DA) field 208. The ToS field 202 is intended to allow an entity to specify the particular service it wants, such as high reliability, fast delivery, accurate delivery, etc., and comprises a number of sub-fields. The sub-fields include a three bit IP precedence (IPP) field 210, three one bit flags (D, T and R) 212, 214 and 216 and two unused bits 218. By setting the various flags, a host may indicate which overall service it cares most about (i.e., Delay, Throughput and Reliability). Although the ToS field 202 was intended to allow layer 3 devices to choose between different links (e.g., a satellite link with high throughput or a leased line with low delay) depending on the service being requested, in practice, most layer 3 devices ignore the contents of the ToS field 202 altogether. Instead, protocols at the transport layer (layer 4) and higher typically negotiate and implement an acceptable level of service. Version 6 of the Internet Protocol (IPv6) similarly defines a traffic class field, which is also intended to be used for defining the type of service to be applied to the corresponding packet.

Recently, a working group of the Internet Engineering Task Force (IETF), which is an independent standards organization, has proposed replacing the ToS field 202 with a one octet differentiated services (DS) field 220. The first six bits of the DS field specify a differentiated services codepoint while the last two bits are unused. Layer 3 devices that are DS compliant apply a particular per-hop forwarding behavior to packets based on the contents of their DS fields 220. Examples of per-hop forwarding behaviors include expedited forwarding and assured forwarding. The DS field 220 is typically loaded by DS compliant intermediate devices located at the border of a DS domain, which is a set of DS compliant intermediate devices under common network administration. Thereafter, interior DS compliant devices along the path simply apply the assigned forwarding behavior to the packet.

Although layer 3 devices, like their layer 2 counterparts, typically have multiple priority queues per port or interface, layer 3 devices often apply scheduling patterns that are more sophisticated than simple Priority Queuing. For example, some layer 3 devices forward one packet from each queue in a round robin fashion. Another approach, referred to as fair queuing, simulates a byte-by-byte round robin to avoid allocating more bandwidth to sources who transmit large packets than to those who only send small packets. Another approach, called Weighted Fair Queuing (WFQ), allocates more bandwidth to specific traffic flows or sources, such as file servers, based on source IP address, Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) source port, etc.

Some networking software, including the Internetwork Operating System (IOS) from Cisco Systems, Inc., support the creation access control lists or filters, which are typically used to prevent certain traffic from entering or exiting a network. In particular, certain layer 3 devices utilize access lists to control whether routed packets should be forwarded or filtered (i.e., dropped) by the device based on certain predefined criteria. When a packet is received by such a device, it is tested against each of the criteria statements of the corresponding list. If a match is found, the packet is either forwarded or dropped as provided by the list. The criteria may be source address, destination address, or upper-layer application based on their TCP/UDP port numbers. For example, an access list may allow e-mail to be forwarded but cause all Telnet traffic to be dropped. Access lists may be established for both inbound and outbound traffic and are most commonly configured at layer 3 devices located at the border of a network (i.e., gateways or firewalls) to provide security to the network.

Congestion Control

Congestion typically refers to the presence of too many packets in a subnet or a portion of a network, thereby degrading the network's performance. Congestion occurs when the network devices are unable to keep up with an increase in traffic. As described above, a layer 3 device typically has one or more priority queues associated with each interface. As packets are received, they are added to the appropriate priority queue for forwarding. Nevertheless, if packets are added to the queue faster than they can be forwarded, the queue will eventually be filled forcing the device to drop any additional packets for that queue. The dropping of packets when a queue is full is referred to as tail drop. The point at which tail drop occurs, moreover, may be configured to something less than the capacity of the queue.

Since tail dropping discards every packet over the queue limit, it often affects multiple upper layer applications simultaneously. Furthermore, many upper layer applications, such as TCP, re-send messages if no acknowledgments are received. Thus, the presence of tail dropping can cause global synchronization among upper layer applications, significantly exacerbating the congestion problem. To avoid global synchronization, some layer 3 devices use Random Early Detection (RED), which selectively drops packets when congestion first begins to appear. By dropping some packets early before the priority queue is full, RED avoids dropping large numbers of packets all at once. In particular, when a calculated average queue depth exceeds a minimum threshold, the device begins dropping packets. The rate at which packets are dropped increases linearly as a function of a probability constant. When a maximum threshold is reached, all additional packets are dropped. An extension to RED is Weighted Random Early Detection (WRED), which applies different thresholds and probability constants to packets associated with different traffic flows. Thus, WRED allows standard traffic to be dropped more frequently than premium traffic during periods of congestion.

Service Level Agreements

To interconnect dispersed computer networks, many organizations rely on the infrastructure and facilities of service providers. For example, an organization may lease a number of T1 lines to interconnect various LANs. These organizations typically enter into service level agreements with the service providers, which include one or more traffic specifiers. These traffic specifiers may place limits on the amount of resources that the subscribing organization will consume for a given charge. For example, a user may agree not to send traffic that exceeds a certain bandwidth (e.g., 1 Mb/s). Traffic entering the service provider's network is monitored (i.e., "policed") to ensure that it complies with the relevant traffic specifiers and is thus "in-profile". Traffic that exceeds a traffic specifier (i.e., traffic that is "out-of-profile") may be dealt with in a number of ways. For example, the exceeding traffic may be dropped or shaped. With shaping, the out-of-profile traffic is temporarily stored until the demand drops below the threshold. Another option is to mark the traffic as exceeding the traffic specifier, but nonetheless allow it to proceed through the network. If there is congestion, an intermediate device may drop this "marked" or down graded traffic first in an effort to relieve the congestion. Another option is to change the accounting actions for this out-of-profile traffic (i.e., charge the user a higher rate).

Allocation of Network Resources

As shown, computer networks include numerous services and resources for use in moving traffic around the network. For example, different network links, such as Fast Ethernet, Asynchronous Transfer Mode (ATM) channels, network tunnels, satellite links, etc., offer unique speed and bandwidth capabilities. Particular intermediate devices also include specific resources or services, such as number of priority queues, filter settings, availability of different queue selection strategies, congestion control algorithms, etc. Nonetheless, these types of resources or services are highly device-specific. That is, most computer networks include intermediate devices manufactured by many different vendors, employing different hardware platforms and software solutions. Even intermediate devices from the same vendor may be running different software versions and thus provide different functionality. Thus, there is no consistency of resources at each of the intermediate devices and, therefore, it is generally not possible to simply select a single set of parameters for use in configuring all of them.

In addition, the allocation of network resources and services is becoming an important issue to network administrators and service providers as greater demands are being placed on their networks. Nonetheless, at the present time, there are few if any techniques available for applying traffic management policies across a network. Instead, the allocation of network resources and services is typically achieved by manually configuring the interfaces of each intermediate device. For example, to the extent there are parameters associated with a particular queuing strategy available at a given intermediate device (e.g., queue length for tail drop and minimum, maximum and mark probability for RED), these parameters must be set device-by-device by the network administrator. This is a time consuming and error prone solution. In addition, there are few if any tools currently available to network administrators suggesting how various network resources and services might be coherently allocated in order to implement any general policies. Accordingly, the ability to allocate network services and resources to implement network-wide quality of service policies is difficult and time-consuming.

SUMMARY OF THE INVENTION

It is an object of the present invention to provide a method and apparatus for applying high-level quality of service policies.

It is a further object of the present invention to provide a method and apparatus for translating high-level policies into a form that may be understood and applied by numerous dissimilar network devices.

It is a further object of the present invention to classify data traffic upon its entering a given network domain and to manage that traffic based on its classification.

Briefly, the invention relates to a method and apparatus for implementing high-level policies within a computer network having multiple, dissimilar network devices. The high-level policies, which are generally device-independent, are selected by a network administrator and translated by one or more policy servers into a set of rules that can be applied by specific network devices. In particular, a network administrator first selects an overall traffic template for a given network domain and may assign various applications and/or users to the corresponding traffic types of the template. The network administrator may also select or define one or more location-specific policies. As information is added to the template and the location-specific policies are defined, one or more corresponding data structures may be up-dated. The selected traffic template, location-specific policies and data structures are received at one or more policy servers within the network domain. Each policy server translates the high-level policies inherent in the selected traffic template, location-specific policies and data structures into a set of rules and may combine several related rules into a single transaction. Upon initialization, intermediate devices request traffic management information from the one or more policy servers. The policy server replies with a particular set of transactions and rules that are utilized by the intermediate devices for traffic management decisions. By propagating these rules across the network domain, each of the dissimilar intermediate devices can configure its corresponding traffic management components and mechanisms to operate in such a manner as to implement the high-level policies selected by the network administrator.

More specifically, a particular differentiated service (DS) codepoint is preferably assigned to each traffic type of the selected traffic template, based on the overall priority established by the network administrator. The DS codepoint essentially sets the overall treatment of the corresponding traffic type within the network domain. A set of classification rules are then generated by the policy server instructing intermediate devices to associate particular traffic types with their corresponding DS codepoints. For example, the classification rules may direct intermediate devices to load a particular DS codepoint within the DS field of received Internet Protocol (IP) messages, depending on the type of IP message (e.g., all traffic associated with a stock exchange application). Devices that are not DS-compliant may receive classification rules instructing them to load a given value within type_of_service or user_priority fields of received packets or frames. The classification rules, which may include one or more access control lists, are preferably provided to all intermediate devices located at the boundary of the network domain so that traffic can be associated with its corresponding DS codepoint as soon as it enters the domain. Classification rules may also be used to associate Quality of Service (QoS) labels to specific traffic types. QoS labels are also used by intermediate devices in making traffic management decisions, although, unlike the DS codepoints which are generally present in messages traveling the network, QoS labels are only associated with messages while they remain within the intermediate device. Classification rules may also be used to assign DS codepoints and/or QoS labels to data traffic generated within the network domain from un-trusted sources.

To implement specific traffic management policies or treatments, the policy server also defines a plurality of behavioral rules that basically instruct the intermediate devices how to manage data traffic that has been associated with a particular DS codepoint, QoS label, type of service and/or user priority value. For example, a behavioral rule may instruct the intermediate devices to place all messages associated with a particular DS codepoint (e.g., data frames from a stock exchange application or from a corporate executive) in a high priority queue. To implement traffic management policies that are independent of DS codepoints and/or QoS labels, the policy servers preferably generate one or more configuration rules. Configuration rules generally instruct intermediate devices how to set-up their various traffic management components or mechanisms. For example, a configuration rule may contain a list of congestion algorithms in descending order of preference. Upon receipt of the configuration rule, an intermediate device examines the list and preferably adopts the first congestion algorithm that it supports.

In the preferred embodiment, the policy servers and intermediate devices utilize an extension to the Common Open Policy Service (COPS) protocol to exchange messages. More specifically, an intermediate device sends a Query Configuration message to the policy server that contains specific information about itself, such as the number and type of interfaces, whether the device is at a boundary of the intermediate domain and/or whether its interfaces are coupled to trusted or un-trusted devices. This device-specific information may be loaded in the query message as COPS objects. In response, the policy server selects a particular set of transactions or rules responsive to the device-specific information and provides them to the intermediate device. Preferably, the transactions and rules are similarly embedded as COPS objects in response messages. As described above, the intermediate device reviews these transactions and rules and implements those rules which are compatible with its particular traffic management components and mechanisms.

BRIEF DESCRIPTION OF THE DRAWINGS

The above and further advantages of the invention may be better understood by referring to the following description in conjunction with the accompanying drawings, in which:

FIG. 1, previously discussed, is a block diagram of a prior art frame;

FIG. 2, previously discussed, is a block diagram of a portion of a prior art Internet Protocol (IP) header;

FIG. 3 is a highly schematic, partial diagram of a computer network;

FIG. 4 is a highly schematic, partial block diagram of a policy server in accordance with the present invention;

FIG. 5 is a highly schematic, partial block diagram of an intermediate device in accordance with the present invention;

FIG. 6 is a preferred traffic template that may be selected by a network administrator; and

FIGS. 7A–7F are block diagrams of data structures associated with the template of FIG. 6.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

FIG. 3 is a highly schematic block diagram of a computer network **300**. The network **300** may be segregated into one or more network domains by a network administrator, such as network domains **302** and **304**, as described below. The network **300** includes a plurality of entities, such as end stations and servers, interconnected by a plurality of intermediate devices, such as bridges, switches and routers. In particular, network **300** includes a plurality of switches **306–314** and routers **316–318** interconnected by a number of links **320***a* to **320***f*, which may be high-speed point-to-point or shared links. Each domain **302** and **304**, moreover, includes at least one policy server **322** that is preferably connected to one or more intermediate devices, such as switch **308** and routers **316–318**, by links **324***a*–**324***c*. The network **300** also includes one or more repositories, such as repository **326**, that is preferably connected to each policy server **322** by a link **328**. The repository **326** may be an organization-based directory server.

The network **300** may further include one or more Dynamic Host Configuration Protocol (DHCP) servers, such as server **329**, that is also coupled to the policy server **322** either directly or indirectly. DHCP, which is defined at Request for Comments (RFC) 2131, is built upon a client-server model, where DHCP servers allocate IP addresses and deliver network configuration parameters to DHCP clients (e.g., hosts or end stations). Because IP addresses can be a scarce resource in some computer networks, DHCP servers assign them only for limited periods of time (referred to as a lease). Once a lease has expired, the corresponding IP address may be re-assigned to another host.

Attached to the switches **306–314** and routers **316–318** are a plurality of end stations **330–346** and servers **348–352**, which may be file servers, print servers, etc. In particular, four end stations **330–336** are connected to switch **306**, one end station **338** and one server **348** are connected to switch **308**, two end stations **340** and **342** are connected to switch **310**, one server **350** is connected to router **318**, two end stations **344** and **346** are connected to switch **312**, and one server **352** is connected to switch **314**.

Software entities (not shown) executing on the various end stations **330–346** and servers **348-352** typically communicate with each other by exchanging discrete packets or frames of data according to predefined protocols, such as the Transmission Control Protocol/Internet Protocol (TCP/IP), the Internet Packet Exchange (IPX) protocol, the AppleTalk protocol, the DECNet protocol or NetBIOS Extended User Interface (NetBEUI). In this context, a protocol consists of a set of rules defining how the entities interact with each other. Data transmission over the network **300** consists of generating data in a sending process executing on a first end station, passing that data down through the layers of a protocol stack where the data are sequentially formatted for delivery over the links as bits. Those frame bits are then received at the destination station where they are re-assembled and passed up the protocol stack to a receiving process. Each layer of the protocol stack typically adds information (in the form of a header) to the data generated by the upper layer as the data descends the stack. At the destination station, these headers are stripped off one-by-one as the frame propagates up the layers of the stack until it arrives at the receiving process.

Preferably, routers **316–318** are layer **3** intermediate devices and thus operate at the internetwork layer of the communication protocol stack implemented within the network **300**. For example, routers **316–318** preferably include an Internet Protocol (IP) software layer, as defined by the well-known TCP/IP Reference Model. Routers **316–318** implement network services such as route processing, path determination and path switching functions. Switches **306–314** may be layer **2** intermediate devices and thus operate at the data link layer of the corresponding communication protocol stack. Switches **306–314** provide basic bridging functions including filtering of data traffic by medium access control (MAC) address, "learning" of a MAC address based upon a source MAC address of a frame and forwarding of the frame based upon a destination MAC address or route information field (RIF). Switches **306–314** may further provide certain path switching and forwarding decision capabilities normally only associated with routers.

In the illustrated embodiment, the switches **306–314** and routers **316–318** are computers having transmitting and receiving circuitry and components, including network interface cards (NICs) establishing physical ports, for exchanging data frames. The switches 306–314 and routers 316–318 further comprise programmable processing elements, which may contain software programs pertaining to the methods described herein. Other computer readable media may also be used to store the program instructions. In addition, associated with each port or physical network connection is one or more logical connections or interfaces defined by the IP software layer.

The terms router or layer 3 intermediate device as used herein are intended broadly to cover any intermediate device operating primarily at the internetwork layer, including, without limitation, routers as defined by Request for Comments (RFC) 1812 from the Internet Engineering Task Force (IETF), intermediate devices that are only partially compliant with RFC 1812, intermediate devices that provide additional functionality, such as Virtual Local Area Network (VLAN) support, IEEE 802.1Q support and/or IEEE 802.1D support, etc. The terms switch and layer 2 intermediate device are also intended to broadly cover any intermediate device operating primarily at the data link layer, including, without limitation, devices that are fully or partially compliant with the IEEE 802.1D standard and intermediate devices that provide additional functionality, such as Virtual Local Area Network (VLAN) support, IEEE 802.1Q support and/or IEEE 802.1p support, Asynchronous Transfer Mode (ATM) switches, Frame Relay switches, etc.

It should be understood that the network configuration 300 of FIG. 3 is for illustrative purposes only and that the present invention will operate with other, possibly far more complex, network topologies. It should be further understood that the repository may be indirectly connected to the policy servers (e.g., through one or more intermediate devices).

As described above, computer networks often include intermediate devices from many different vendors or, even if from the same vendor, having different hardware architectures or executing different versions of software. Accordingly, these intermediate devices provide many different features and options. For example, a first switch may provide only 2 priority queues per port, whereas a second switch may provide 8 priority queues per port. With regard to congestion algorithms and techniques, some intermediate devices may only support tail dropping, while others may be selectively configured to provide random early detection (RED). Thus, it is extremely difficult for a network administrator to configure all of the intermediate devices in accordance with a single, uniform traffic management plan. As result, network-wide quality of service is generally not available. As described herein, the present invention provides a method and apparatus for allowing network administrators to apply high-level traffic management policies that attempt to impose such a uniform plan, despite the presence of dissimilar intermediate devices in their networks. The traffic management policies, moreover, may be automatically propagated to and implemented by the various intermediate devices.

FIG. 4 is a highly schematic, partial block diagram of policy server 322 in accordance with the preferred embodiment of the present invention. The policy server 322 is comprised of several components, including a policy translator 410 having one or more storage devices 412a–412c. Policy server 322 also includes a policy validation tool (PVT) 413 and a policy rule generating engine 414 that are each in communication with the policy translator 410, a device-specific filter entity 416 and a communication engine 418. As shown, the device-specific filter entity 416 communicates with both the policy rule generating engine 414 and the communication engine 418. The communication engine 418, moreover, is preferably configured to exchange messages with the intermediate devices (e.g., switches 306–314 and routers 316–318) of network 300. That is, communication engine 418 is connected to or includes conventional circuitry for transmitting and receiving messages across network links, such as links 324a–324c.

A server suitable for use as policy server 322 is any Intel/Windows NT® or Unix-based platform.

FIG. 5 is a partial block diagram of an intermediate device, such as router 318, in accordance with the preferred embodiment of the present invention. Router 318 preferably includes a communication engine 510 that is coupled to a traffic management controller 512. The communication engine 510 is configured to exchange messages with the policy server 322. That is, communication engine 510, like the communication engine 418 at policy server 322, is similarly connected to or includes conventional circuitry for transmitting and receiving messages across the network 300. The traffic management controller 512, which includes a policy rule decoder 514, is coupled to several components and mechanisms. In particular, traffic management controller 512 is coupled to a packet/frame classifier 516, a traffic conditioner entity 518, a queue selector/mapping entity 520 and a scheduler 522. The traffic conditioner 518 also includes several sub-components, including one or more metering entities 524, one or more marker entities 526 and one or more shaper/dropper entities 528. The queue selector/mapping entity 520 and scheduler 518 operate on the various queues established by router 318 for its ports and/or interfaces, such as queues 530a–530e corresponding to an interface 532.

Creation of QoS Domains and Selection of High-Level Policies

First, the network administrator preferably identifies various regions of his or her computer network 300 to which he or she wishes to have different, high-level traffic management polices applied. The identification of such regions may depend on any number of factors, such as geographic location, business unit (e.g., engineering, marketing or administrative), anticipated network demands, etc. The network administrator preferably defines a separate Quality of Service (QoS) or network domain for each region and assigns a primary policy server (e.g., policy server 322) to each QoS domain (e.g., domain 302). Thus, a QoS domain is basically a logical set of entities and intermediate devices defined by the network administrator. As described below, the primary policy server is responsible for propagating the high-level traffic management polices to the intermediate devices within its QoS domain.

It should be understood that the policy server 322 may, but need not, be physically located within its QoS domain. It should be further understood that back-up or standby policy servers may also be assigned to the QoS domains should any primary policy server fail.

In addition, the boundaries of the network domains 302, 304 may be established so as to only include trusted devices. A "trusted device" is an entity (e.g., an end station or server) which is considered to correctly classify the packets that it sources and to keep its transmission of packets in-profile (i.e., within the bounds of the traffic specifiers of any applicable service level agreements). A packet is classified by loading its user_priority field 114, ToS fields 202 and/or DS field 220 with a particular value or codepoint. Similarly, an "un-trusted device" is an end station or server which is not assumed to correctly classify its own packets and/or maintain its flow of traffic within all applicable traffic specifiers. Packets from an un-trusted device must be examined and reclassified as necessary. Additionally, the flow from un-trusted devices must be policed. In a similar manner, the ports of an intermediate device that are coupled to one or more un-trusted devices are referred to as "untrusted ports", whereas ports coupled to only trusted devices are "trusted ports".

Once the QoS domains have been defined, the network administrator preferably proceeds to select the high-level, device-independent traffic management policies that are to be implemented within each domain. First, the network administrator selects an overall traffic template that establishes the different traffic types that are to be supported within the respective QoS domain. In particular, the network administrator may select one of several available traffic templates. An exemplary traffic template may be the traffic type list established by the IEEE in the 802.1p standard, which defines the following traffic types: best effort, background, excellent effort, controlled load, video, voice and network control, as described above. Other traffic templates include a financial template, a manufacturing template and a university or education template.

FIG. 6 is a highly schematic representation of a financial template **610** for use by a network administrator in accordance with the present invention. As shown, the financial template **610** includes a first column **612** listing a plurality of available traffic types corresponding to the financial template **610**. The available traffic types include best effort, background, CEO best effort, voice, business applications, stock exchange applications, 500 kb/s video conference, 2 Mb/s video conference and network control. A second column **614** identifies a particular differentiated service (DS) value corresponding to each traffic type. The DS codepoint establishes the overall treatment that is to be assigned to the corresponding traffic type within the respective QoS domain **302**. To fit within the first six bits of DS field **220**, DS codepoints are in the range of 0–63. As described below, the DS codepoints may also be used by intermediate devices in loading the user_priority and/or ToS fields **114**, **202** with corresponding values during classification.

A third column **616** identifies the network users who may take advantage of the various traffic types. For example, the network administrator may decide that any network user may utilize the best effort, background, voice, 500 kb/s video, 2 Mb/s video and network control traffic types. However, only the chief executive officer (CEO) may take advantage of the CEO best effort traffic type and only network users from the marketing, administrative, executive, financial analysis and financial planning departments may utilize the business applications traffic type. Similarly, only network users from the financial analysis, financial planning and trading departments may use the stock exchange applications traffic type. A fourth column **618** identifies the application programs corresponding to each traffic type. For example, available business applications may include a spreadsheet application, a word processor application, or any of the well-known and commercially available business applications from SAP AG of Walldorf, German or PeopleSoft, Inc. of Pleasanton, Calif. A stock exchange application may be TIB from TIBCO Inc. of Palo Alto, Calif. The identification of network users in column **616** and the application programs in column **618** are preferably entered by the network administrator. The network administrator may rely on default DS codepoints in column **614** or may change these values as desired. The traffic types and DS codepoints for a given template are preferably derived from empirical studies and analysis of the computer network operations and usages of such industries and organizations.

In order to select the desired template and enter the requested information, the network administrator may interact with various windows of a graphical user interface (GUI). These windows, for example, may present fields, such as the entries for columns **616** and **618**, having pull-down menus that request information from the network administrator. The information may be entered by the network administrator using a mouse or keyboard in a conventional manner. The user interface, moreover, is preferably similar in operation to the Cisco Works Windows interface (for configuring router interfaces) or the VlanDirector interface of the Cisco Works for Switched Internetworks (CWSI) interface (for configuring VLANs), both from Cisco Systems, Inc.

It should be understood that other means of associating traffic types to users, applications and DS codepoints, besides traffic templates, may be employed. It should also be understood that network administrators may select different traffic templates or adjust their parameters for different times of day or for emergency situations.

Next, the network administrator defines any location-specific policies. For example, the network administrator may specify that intermediate devices located at the border of the QoS domain should only accept traffic that belongs to a specific group, such a company employees, department members, etc. Any traffic which does not belong to the group should be dropped. The network administrator may also define one or more lists of global parameters that are to be utilized throughout the QoS domain. An example of a global parameter list is a prioritized list of queue scheduling algorithms from first choice to last choice, such as WFQ, WRR and Priority Queuing (PQ). Other examples of global parameter lists include congestion algorithms (e.g., RED over tail dropping), enabling multi-link Point-to-Point Protocol (PPP) fragmentation, if available, and enabling Virtual Circuit (VC) merging, if available.

Associated with the template **610** are one or more data structures and, as information is entered into the template **610**, these data structures are preferably updated accordingly. As described below, these data structures are used to generate the traffic management rules implemented by the intermediate devices. FIGS. 7A–7E are block diagrams of exemplary data structures associated with template **610**. In particular, FIG. 7A is a network user table **710** that maps users identified in column **616** of template **610** with actual user names and/or IP addresses and masks. User table **710** preferably includes a user column **712**, a name column **714**, an IP address column **716**, an IP mask column **718** and a plurality of rows such that the intersection of a column and row defines a table entry. As information is entered in column **616** of template **610**, corresponding entries are made in the user column **712** of table **710**. As described below, information for columns **714**, **716** and **718** is subsequently added by the policy server **322**. FIG. 7B is an application table **730** that maps the applications programs entered on the financial template **610** to their network protocol, such as the Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) port numbers. In particular, application table **730** preferably includes a first column **732** listing the application programs identified in the selected template **610** and a second column **734** that identifies both the transport protocol and the port number for each corresponding application program.

FIG. 7C is a classifier table 740 that maps DS codepoints, including those specified by the network administrator in the selected template 610, with corresponding values for use in classifying or shaping traffic within the corresponding QoS domain 302, as described herein. In particular, each available DS codepoint (0–63), which is loaded in a first column 742, is preferably mapped to a DS mark down value contained in a second column 744, a User Priority value contained in a third column 746 and a Type of Service (ToS) value contained in a fourth column 748. Preferably, table 740 is preconfigured with a set of default values corresponding to the selected template 610. The network administrator may, however, access table 740 while establishing the high level policies and modify these values.

FIGS. 7D–E are exemplary queue/threshold assignment tables that map DS codepoints to queues and thresholds depending on the number of queues and thresholds that are available at a given interface. For example, FIG. 7D is a first queue/threshold assignment table 760 for an interface supporting two queues and two thresholds per queue. As shown, table 760 includes a column 762, 764 for each queue and a row 766, 768 for each threshold. At the intersection of each column and row is cell 770*a–d* that contains the set of DS codepoints for the corresponding queue/threshold combination. For example, cell 770*a* identifies the set of DS codepoints (e.g., 0, 4, 8, etc.) to be assigned queue 1 and threshold 1. Similarly, cell 770*d* identifies the set of DS codepoints (e.g., 3, 7, 11, 62, 63, etc.) to be assigned queue 2 and threshold 2. FIG. 7E is a second queue/threshold assignment table 774 for an interface supporting 2 queues and 4 thresholds. Accordingly, table 774 includes two columns 776, 778 (one for each queue) and four rows 780–783 (one for each threshold) whose intersections define a plurality of cells 784*a–h*. The cells 784*a–h* contain the set of DS codepoints for the corresponding queue/threshold combination.

FIG. 7F illustrates a third queue/threshold assignment table 788 for interfaces supporting 5 queues and 2 thresholds. Thus, table 788 includes 5 columns 790–794 and 2 rows 795, 796 whose intersections define a plurality of cells 798*a–j*. As described above, each cell 798*a–j* includes a set of DS codepoints for the corresponding queue/threshold combination. For example, cell 798*a* includes DS codepoints 0, 10, etc., while cell 798*g* includes DS codepoints 6, 19, 60, etc.

It should be understood that a queue/threshold assignment is preferably generated for each number of queue/threshold combinations supported by the interfaces in the network.

It should be further understood that tables 760, 774 and 788 may only assign a subset of DS codepoints to queues and thresholds, rather than all DS codepoints. For example, DS codepoints may be segregated into standardized and private classes or codepoints. Standardized codepoints are assigned particular per hop behaviors by the IETF such as expedited or assured forwarding. Private codepoints may be associated with any treatment on an implementation-by-implementation basis. The present invention preferably maintains the associated behaviors of any standardized codepoints.

Generation of Policy Rules Based on the Selected High-Level Policies

Referring to FIG. 4, these high-level policies, including the financial template 610 (FIG. 6), data structures 710, 730, 740, 760, 774 and 788 (FIGS. 7A–F) and location-specific policies, if any, are then provided to the policy server 322. In particular, the information is received at the policy translator component 410, as shown by arrow 420. Policy translator 410 examines the high-level policies and corresponding data structures and may perform certain initial processing. For example, to the extent the user table 710 lists individual or group network users by title or department, the policy translator 410 may identify the actual users and obtain their IP addresses and/or corresponding subnet masks. For example, by accessing the repository 326 and/or other information resources, such as DHCP server 329, the policy translator 410 may enter additional information in table 710. In particular, the policy translator 410 may query the repository 326 or DHCP server 329 to obtain the CEO's name, IP address and IP mask. This information may then be inserted in the corresponding entries of user table 710. Similar information, where appropriate, may be obtained for groups, such as the marketing, administrative and executive departments, from repository 326 or DHCP server 329, and entered into user table 710. The policy translator 410 may employ a conventional database query-response application such as SQL and the Light weight Directory Access Protocol (LDAP) to communicate with the repository 326 and DHCP server 329. Alternatively, the policy translator 410 may be pre-configured with such information.

The information for column 732 of the application table 730 may also be obtained and inserted by the policy translator 410. In particular, policy translator 410 may include a database that correlates application programs to transport protocol and port number. Many applications, such as the hyper text transport protocol (HTTP), are assigned specific, fixed TCP/UDP port numbers, such as port 80, in accordance with Request for Comments (RFC) 1700. This information may be stored by the policy translator 410 in a conventional manner. Although RFC 1700 provides fixed port numbers for hundreds of applications, there are still many applications that do not have predefined, fixed port numbers. The port numbers utilized by these application are typically selected dynamically by the instances of the application program executing at the sending and receiving devices at the time the respective communication session is established.

To identify these dynamically selected port numbers, the intermediate devices may perform a stateful inspection of received packets for any given communication session. This stateful inspection will reveal the port numbers selected by the corresponding entities. A suitable method for performing such stateful inspections is the Context Based Access Control feature of the Internetwork Operating System (IOS) from Cisco Systems, Inc. For some application programs, corresponding software modules may exist for identifying the selected port numbers for any given session of that application program. For example, software module @h245.voice.inspect is used to identify the port numbers selected by instances of H323.voice applications. Policy translator 410 may be configured with the identity of these modules for insertion in the appropriate entry of application table 710.

It should be understood that these data structures (e.g., tables 710, 730, 740, 760, 774 and 788) may be stored by policy translator 410 at its storage devices 412*a*–412*c*. It should be further understood that the policy generator 410 may also generate and store additional data structures in response to the high-level policies selected by the network administrator.

As tables 710, 730, 740, 760, 774 and 788 are loaded and/or up-dated, the policy rule generating engine 414 accesses this information and creates one or more rules that can be transmitted to the intermediate devices within the respective QoS domain 302. These rules, moreover, which may include one or more access control lists, are in a format that is both readable and executable by the intermediate devices, as described below.

First, the policy rule generating engine 414 creates a set of classification rules. Classification rules are generally utilized by intermediate devices to assign a given treatment to network traffic based on certain criteria, such as source or destination address, protocol, port number, application program, etc. In the preferred embodiment, classification rules, which include one or more objects, are applied at specific locations (e.g., an interface or group of interfaces coupled to un-trusted devices) or at intermediate devices located at the boundary of the QoS domain 302. Location-specific classification rules preferably have the following format:

<location><direction><acl><rmo><Classification_Decision_Rule>

where, the "location" object identifies a particular interface, interface type or role as described below, the "direction" object refers to whether the rule is to be applied to packets at the input, output or both portions of the interface(s), the "access control list" (acl) object contains a list of criteria statements to be applied to the packets and the "rule management object" (rmo) instructs the intermediate device how to respond if conflicting actions are returned and the "Classification_Decision_Rule" object is the actual rule or rules being implemented to packets matching the acl object. Although the rmo preferably instructs the intermediate device to select the best match, other tie-breaking solutions may be presented. The second format of a classification rule, which is used with intermediate devices located at the boundary of the QoS domain 302, appears as follows.

<acl><rmo><Classification_Decision_Rule>

In addition, the acl object may have one of two formats.

(1) <destination IP address or destination IP mask><source IP address or source IP mask><protocol><source and/or destination port numbers> or

(2) <destination or source MAC address>

In the preferred embodiment, classification rules are used for one of three primary purposes: (1) assigning a DS codepoint to packets, (2) assigning a QoS label to packets while they are processed within an intermediate device or (3) instructing an intermediate device to shape, mark and/or drop out-of-profile traffic. For example, a classification rule may be used at the border intermediate devices instructing them to drop packets with a given source IP address or IP mask. A classification rule may also be used to assign a given DS codepoint to all traffic associated with a given IP mask (e.g., all traffic from the marketing department) or all traffic associated with a given port (e.g., port 23 for Telnet).

As described above, only sixty-four DS codepoints are supported by the DS field 220. To extend the concept of packet-specific differentiated services beyond sixty-four options, the present invention also utilizes Quality of Service (QoS) labels. A QoS label is a name string of any length (e.g., an integer, an alphanumeric string, etc.) that may be associated with a packet while it remains internal to the intermediate device. Classification rules may also be used to assign QoS labels to packets based on their source or destination address, protocol, application, etc. As described below, intermediate devices maintain a mapping of QoS labels to traffic types and to the corresponding action to be taken or service to be provided.

Next, the policy rule generating engine 414 creates a set of behavioral rules, which are utilized to instruct intermediate devices how to treat data traffic assigned a particular DS codepoint and/or QoS label by the classification rules. Behavioral rules also include one or more objects and are preferably applied at all compliant intermediate devices within the QoS domain 302. Behavioral rules may be location-specific or location-independent. The preferred format of a location-specific behavioral rule is as follows.

<location><direction><label_Test><Behavioral_Rule_Decision>

where the "label_Test" object may be <dsc_Test> (e.g., DS codepoint=N, where N is some number, such as "32") or <QoS_Label_Test> (e.g., QoS Label=N). The preferred format of a location-independent behavioral rule is as follows.

<label_Test><Behavioral_Rule_Decision>

By applying the label_Test to each packet, the intermediate device determines whether the corresponding Behavioral_Rule_Decision should be applied. The Behavioral_Rule_Decision object preferably implements one or more of five possible decisions: select queue, select queue threshold, set the User_Priority field 114, set the IPP sub-field 210 or shape, mark and/or drop packets satisfying the label_Test object. For example, a behavioral rule may instruct the intermediate devices to set to "6" both the User_Priority field 114 and the IPP sub-field 210 of all frames or packets whose DS codepoint is "61". Similarly, another behavioral rule may instruct intermediate devices to place all messages whose DS codepoint is "32" (e.g., data frames from a stock exchange application) in a high priority queue. Behavioral rules may similarly specify a particular treatment based on the QoS label, user priority or type of service of a packet or frame, such as fast or reliable service. In response, an intermediate device may select a particular transmission link. Since behavioral rules lack an rmo object, intermediate devices apply all behavioral rules they support, not just the first one. If multiple behavioral rules specify contradictory actions, the last one preferably takes precedence.

To implement traffic management policies that are independent of DS codepoints and/or QoS labels, the policy rule generating engine 414 preferably creates a plurality of configuration rules. In general, configuration rules instruct intermediate devices how to set-up their various traffic management components or mechanisms. Configuration rules also have a location-specific and location-independent format which are preferably as follows.

<location><direction><Configuration_Rule_Decision>

<Configuration_Rule_Decision>

The "Configuration_Rule_Decision" object may be used to specify certain global parameters or algorithm parameters. For example, the Configuration_Rule_Decision object of a given configuration rule may contain a list of congestion algorithms in descending order of preference, such as WFQ, WRR, PQ and none. In addition, if an intermediate device uses tail drop and supports four different drop thresholds per queue, a configuration rule may set the four thresholds (e.g., at 50%, 80%, 95% and 100% of the buffer limit) and assign a name to each threshold. Similarly, if an intermediate device supports WRED, a configuration rule may be used to set the minimum threshold, maximum threshold and probability constant for each weight. Also, for

WRR, a configuration rule may assign the weights to the various queues.

The rule generating engine 414 may also combine several related rules into a transaction. More specifically, rules that are meaningful only if applied simultaneously and which may cause transient misconfigurations if implemented one at a time, are combined into a transaction. For example, a network administrator may want to log as well as drop all attempts to access a subnetwork or LAN by a known hacker. Rather than issue a separate rule that only provides for logging and a subsequent rule for dropping, which might result in transitory access by the hacker, these two rules (log and drop) are preferably combined into a single transaction. A "transaction start" object is preferably used to indicate the start of a set of rules forming a transaction and a "transaction end" object indicates the end. As described below, the rules and transactions are accessible by the device-specific filter entity 416 which collects relevant rules for transmission to the intermediate devices.

To generate the particular rules for a given QoS domain, the policy rule generating engine 414 preferably performs a conventional algorithmic transformation on the corresponding data structures (e.g., tables 710, 730, 740, 760 and 770). This algorithmic transformation converts the information from the data structures into the necessary access control list objects and classification, behavioral and configuration rule objects of the corresponding rules. Such algorithmic transformations are well-known to those skilled in the art. The objects comprising the various rules, including the rule objects themselves, may be defined using Abstract Syntax Notation One (ASN.1) which is well-known to those skilled in the art.

The policy translator 410 also interfaces with the policy validation tool (PVT) 413 to identify any conflicting policies. That is, the PVT 413 examines the high-level policies and performs a conflict check. In particular, the PVT 413 determines whether the policies ascribe conflicting treatments to the same traffic. For example, two polices may call for different shaping or marking to be applied to the same traffic stream. Another policy may be incomplete by failing to specify a requisite condition. All conflicts detected by the PVT 413 are reported to the policy translator 410. The PVT 413 may also determine whether sufficient network resources exist to implement the policies. For example, a policy may require at least one network path having 3 or more queues at each intermediate device along the path. If no such path exists, the PVT 413 preferably reports this condition to the policy translator 410.

Propagation of the Policy Rules to Intermediate Devices

In operation, intermediate devices within a QoS domain will request traffic management information from the local policy server. This information will then be utilized by the intermediate devices in setting their resources and in making traffic management decisions. In the preferred embodiment, the policy server and intermediate devices utilize an extension to the Common Open Policy Service (COPS) client-server communication protocol. In particular, the policy server and the intermediate devices preferably utilize the COPS extension described in *COPS Usage for Differentiated Services*, an Internet Draft Document, dated August 1998, from the Network Working Group of the IETF, which is hereby incorporated by reference in its entirety.

More specifically, referring to FIGS. 4 and 5, upon initialization of router 318, the traffic management controller 512 polls the various components and mechanisms to determine what network resources and services router 318 has to offer. For example, traffic management controller 512 determines that router 318 has five queues 530*a*–530*e* per interface 532. Additionally, traffic management controller 512 may determine that each queue 530*a*–530*e* may support either RED or tail dropping and supports two settable thresholds per queue. The traffic management controller 512 may further identify the roles assigned to one or more of its interfaces.

Roles preferably specify the type or nature of an interface or sub-interface. For example, an interface may be trusted or un-trusted. It may configured to perform policing and shaping of traffic from a subscribing network. It may be a backbone interface and thus multiplex large volumes of traffic to the backbone network or it may be a QoS border interface. An interface may also have more than one role. The particular role or roles of an interface are preferably assigned by a network administrator utilizing a management protocol, such as Simple Network Management Protocol (SNMP) or CiscoWorks from Cisco Systems, Inc., during configuration of the interface. A corresponding flag, label or name may be maintained by the device to identify the various roles of its interfaces. For router 318, the interface coupled to domain 304 may be assigned the role of policing and shaping traffic from subscribing domain 304 in accordance with one or more traffic specifiers.

The assignment of roles facilitates the creation and implementation of network policies. In particular, global policies may be defined that apply to all interfaces regardless of their particular roles. Local policies apply to the role at one specific interface. In other words, policies may be assigned to roles and roles may be assigned to the various interfaces in the network. Thus, by simply changing the role at a given interface, a network administrator ensures that the appropriate network policies are automatically propagated to and implemented by that interface. Each role, moreover, may have a corresponding precedence to resolve any conflicts that might arise at an interface assigned more than one role.

All of this information may be transmitted by the traffic management controller 512 to the communication engine 510 along with an instruction to send to the information to the policy server 322. In response, the communication engine 510 preferably formulates a Configuration Request message that includes the information received from the traffic management controller 512 as a series of objects. The Configuration Request message is then transmitted by the communication engine 510 to the policy server 322.

At the policy server 322, the Configuration Request message is received at the corresponding communication engine 418 and handed to the device-specific filter entity 416. The device-specific filter entity 416 examines the Configuration Request to determine what types of network resources and services are available at router 318 and what roles if any are associated with its interfaces. In particular, the device-specific filter entity 416 determines that router 318 supports both RED and tail dropping, has five queues with two settable thresholds per queue and an interface whose role is to police and shape traffic from a subscribing network. Based on this determination, the device-specific filter entity 416 obtains a particular set of transactions and/or rules from the policy rule generating engine 414 that corresponds to the network services and resources available at router 318. For example, the device-specific filter entity 416 may obtain one or more classification rules instructing router 318 to classify packets from a given source (e.g., domain 304) with a given DS codepoint and/or QoS label. Rules for policing and shaping traffic from domain 304 may also be obtained.

Additionally, the device-specific filter entity 416 may obtain one or more behavioral rules that instruct router 318 to map packets with various DS codepoints to specific queues and thresholds in accordance with the information contained in table **788** (FIG. 7F). More specifically, a first behavioral rule may provide for mapping packets with a DS codepoint of 0, 10, etc. (e.g., DS codepoints corresponding to cell **798***a*) to queue 1 (e.g., queue **530***a*) and the lower threshold. Another behavioral rule may map packets with a DS codepoint of 6, 19, 60, etc. (e.g., DS codepoints corresponding to cell **798***g*) to queue 2 (e.g., queue **530***b*) and the second threshold and so on. Thus, a set of behavioral rules are obtained that will allow router **318** to map various packets based on their DS codepoints to queues **530***a*–**530***e* and corresponding thresholds.

Filter entity **416** may also obtain one or more configuration rules. For example, filter entity **416** may obtain a configuration rule for use in setting the scheduler **522**. In particular, a configuration rule may provide a list of scheduling algorithms in a preferred order (e.g., WFQ, WRR and Priority Queuing). Another configuration rule may provide that Virtual Circuit merging should be applied where available. Filter entity **416** may access the policy rules via a virtual information store, such as the Policy Information Base (PIB) specified in the draft COPS Usage for Differentiated Services document.

Once the device-specific filter entity **416** has obtained a set of transactions or rules for router **318**, it provides them to the communication engine **418** which, in turn, loads them into one or more Decision Messages. These Decision Messages are then transmitted by communication engine **418** to router **318**. Communication engine **510** at router **318** receives the Decision Messages, extracts the rules contained therein and provides them to the traffic management controller **512** where they may be decoded by policy rule decoder **541**. Traffic management controller **512** may also build one or more data structures (such as tables) to store the mappings contained in any received behavioral rules.

It should be understood that intermediate devices learn of the identity of the policy server **322** through any conventional means, such as manual configuration or a device configuration protocol.

Implementation of the Policy Rules at Specific Intermediate Devices

First, traffic management controller **512** proceeds to configure its components and mechanisms in accordance with the instructions contained in the classification rules. For example, to the extent router **318** supports Virtual Circuit merging, this feature is enabled. Similarly, to the extent scheduler **522** can implement WRR and Priority Queuing, traffic management controller **512** configures it to use WRR. As packets are received at router **318**, they are examined by the packet/frame classifier **516** which reports the contents of the packet's User_Priority field **114**, IPP sub-field **210** and/or DS field **220** to the traffic management controller **512**. Packet/frame classifier **516** may also supply other packet header information to the traffic management controller, such as source IP address, port, protocol, etc. In response, the traffic management controller **512** relies on the received behavioral rules to determine in which queue **530***a*–**530***e* the corresponding packet should be placed for forwarding and to instruct the queue selector/mapping entity **520** accordingly. Similarly, router **318** relies on the behavioral rules to determine which packets to mark down and/or drop.

Furthermore, to the extent router **318** policies traffic received from subscribing domain **304**, additional configuration rules may be provided to router **318** for setting its traffic conditioner entity **518**. For example, one or more configuration rules may instruct router **318** to activate its meter entity **524** so as to monitor the traffic arriving from domain **304**. If out-of-profile traffic is to be marked through marker entity **526**, classification rules may be provided for re-setting the DS codepoints of traffic that is out-of-profile based on the information contained in table **740**. Alternatively, if out-of-profile traffic is to be shaped or dropped, other configuration rules may instruct the associated traffic management controller **512** to set the shaper/dropper entity **528** accordingly.

This process is similarly repeated at each of the intermediate devices within the QoS domain **302** that are compliant with the present invention. Depending on the particular network resources and services available at each intermediate device, a different set of rules will be selected by the device-specific filter entity **416**. For example, switch **306** may similarly send a Configuration Request message to policy server **322** and receive a Decision Message. Furthermore, based on the information contained in the Configuration Request message from switch **306**, including the fact that switch **306** is coupled to one or more un-trusted devices, such as end stations **332**–**336**, the device-specific filter entity **416** may obtain one or more classification rules for classifying traffic received from these un-trusted devices. However, since switch **306** may not operate at the network layer, filter entity **416** may obtain classification rules for setting the User_Priority field **114** of packets or frames received on ports coupled to devices **332**–**336**, depending on various parameters of the packets or frames, such as port number, protocol type, etc. Filter entity **416** may also obtain behavioral rules instructing switch **306** how to handle packets based on the user priority value rather than DS codepoint, since switch **306** may not be DS-compliant. Alternatively, policy server **322** may provide one or more classification rules that map User Priority values to DS codepoints so that switch **306** may apply one or more behavioral rules that are dependent on DS codepoints to packets that have a User Priority value.

It should also be understood that less than all of the intermediate devices within a given network may be configured to implement the present invention, although in the preferred embodiment, all of the intermediate devices will be so configured.

The foregoing description has been directed to specific embodiments of this invention. It will be apparent, however, that other variations and modifications may be made to the described embodiments, with the attainment of some or all of their advantages. For example, other client-server communications protocols, besides COPS, may be utilized by the policy server and intermediate devices. In addition, the present invention may also be utilized with other network layer protocols, such as IPv6, whose addresses are 128 bits long. The present invention may also be used to classify other fields of data messages, such as the User Priority field of the Inter-Switch Link (ISL) mechanism from Cisco Systems, Inc. Therefore, it is the object of the appended claims to cover all such variations and modifications as come within the true spirit and scope of the invention.

What is claimed is:

1. A method for implementing high-level, device-independent traffic management policies within a computer network having multiple, dissimilar intermediate network devices, the method comprising the steps of:

selecting one or more high-level policies;

translating the one or more high-level policies into a plurality of executable rules;

receiving a request for traffic management policies from an intermediate device supporting a set of network services;

selecting, in response to the request, one or more rules that are compatible with the network services supported by the intermediate device;

forwarding the selected one or more rules to the intermediate device; and

utilizing the one or more rules to configure the set of network services at the intermediate device to realize the selected high-level policies.

2. The method of claim 1 wherein the rules formulated by the step of translating include at least one of classification, behavioral and configuration rules.

3. The method of claim 2 wherein the step of selecting further includes the step of selecting a predefined traffic template and loading the selected template with user and application information.

4. The method of claim 3 further comprising the step of up-dating one or more data structures associated with the selected template as user and application information is inserted therein.

5. The method of claim 4 wherein at least one classification rule includes an access control list object, a rule management object and a classification decision rule object.

6. The method of claim 5 wherein the at least one classification rule further includes a location object and a direction object.

7. The method of claim 6 wherein at least one behavioral rule includes a label test object and a behavioral rule object.

8. The method of claim 7 wherein the at least one behavioral rule further includes a location object and a direction object.

9. The method of claim 8 wherein at least one configuration rule includes a configuration rule object.

10. The method of claim 9 wherein the at least one configuration rule further includes a location object and a direction object.

11. The method of claim 10 wherein the step of translating includes the step of performing an algorithmic transformation on the one or more data structures to obtain the corresponding classification, behavioral and configuration rules.

12. A policy server for use in implementing high-level, device-independent traffic management policies within a computer network having multiple, dissimilar intermediate network devices and one or more information resources, the policy server comprising:

means for receiving the high-level traffic management policies including one or more corresponding data structures;

a policy translator that is configured to access the one or more information resources for inserting information in the data structures;

a policy rule generating engine coupled to the policy generator and configured to translate the data structures into one or more executable traffic management rules;

a device-specific filter entity coupled to the policy rule generating engine and configured to select a subset of the one or more traffic management rules in response to a request from a respective intermediate network device having particular traffic management resources and services; and

and a communication engine coupled to the device-specific filter entity for exchanging requests from intermediate network devices and selected subsets of the one or more traffic management rules.

13. The policy server of claim 12 wherein the one or more corresponding data structures include a user table that maps individual network users identified in the high-level polices to network addresses and maps network groups to network masks.

14. The policy server of claim 13 wherein the one or more corresponding data structures further include an application table that maps application programs identified in the high-level policies to network protocol and port number.

15. The policy server of claim 14 wherein the high-level traffic management policies are represented by a selected traffic template that maps each of a plurality of traffic types defined by the selected traffic template with at least one of a differentiated service (DS) codepoint, a network user and an application program.

16. The policy server of claim 15 wherein the one or more corresponding data structures further include a queue assignment table that maps DS codepoints to queue numbers.

17. The policy server of claim 16 wherein the one or more corresponding data structures further include a queue threshold table that maps DS codepoints to queue thresholds.

18. The policy server of claim 17 wherein the one or more corresponding data structures further include a priority table that maps DS codepoints to DS mark down values, user priority values and type of service values.

* * * * *

US005473607A

United States Patent [19]

Hausman et al.

[11] **Patent Number: 5,473,607**

[45] **Date of Patent: Dec. 5, 1995**

[54] **PACKET FILTERING FOR DATA NETWORKS**

[75] Inventors: **Richard J. Hausman**, Soquel; **Lazar Birenbaum**, Saratoga, both of Calif.

[73] Assignee: **Grand Junction Networks, Inc.**, Union City, Calif.

[21] Appl. No.: **103,659**

[22] Filed: **Aug. 9, 1993**

[51] **Int. Cl.^{6}** **H04L 12/28**

[52] **U.S. Cl.** **370/85.13**; 370/94.1; 370/92

[58] **Field of Search** 370/60, 60.1, 85.13, 370/85.14, 94.1, 94.2, 94.3, 92, 85.3; 395/159, 118, 400

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,399,531	8/1983	Grande et al.	370/60
4,627,052	12/1986	Homre et al.	370/85.13
4,679,193	7/1987	Jensen et al.	370/94.1
4,891,803	1/1990	Juang et al.	370/60
4,933,937	6/1990	Konishi	370/85.13
5,032,987	7/1991	Broder et al.	364/200
5,210,748	5/1993	Onishi et al.	370/60
5,218,638	6/1993	Matsumoto et al.	380/23
5,247,620	9/1993	Fukuzawa et al.	370/85.13
5,274,631	12/1993	Bhardwaj	370/60

OTHER PUBLICATIONS

Kalpana, Etnerswithc Product Overview, Mar. 1990, pp. 1-20.

Artel, Galactica Stenbridg C/802.3 Application Note, Nov. 1991, pp. 1-26.

Synemetics, Lanplex 5000: Intra-Network Banowidth, 1992, pp. 1-12.

Synemetics, Lanplex 5000 Intelligent Switching Hubs, 1993, pp. 1-9.

Synemetics, Lanplex 5000 Family, 1992. pp. 1-6.

Synemetics, Etheract Express Module, 1991, pp. 1-4.

Alantec, Powerhub Arhitectures, Dec. 1992, pp. 1-6.

Bradner, Scott O., "Ethemet Bridges and Routes", Feb. 1992, pp. 1-10, Data Communications.

Kwok, Conrad K. and Biswanath Mjkherjec, "Cut-Through Bridging for CSMA/CE Local Area Network", Jul. 1990, pp. 938-942, I EEE Transactions on Communications, vol. 38, No. 7.

Primary Examiner—Douglas W. Olms
Assistant Examiner—Chau T. Nguyen
Attorney, Agent, or Firm—Michael J. Hughes

[57] **ABSTRACT**

An improved partial packet filter (**10**) for filtering data packets (**210**) in a computer network (**12**) wherein a candidate field (**413**) of the data packet (**210**) is hashed to a plurality of bit-wise subsets (**636**) each being an independent representation of the candidate field (**413**). Each of the bit-wise subsets (**636**) is compared to a reference hash table (**644**) which has been prepared in a preliminary operation series (**514**). The preliminary operation series (**512**) configures a plurality of target fields (**714**) to set selected memory locations (**312**) in the reference hash table (**644**).

17 Claims, 4 Drawing Sheets

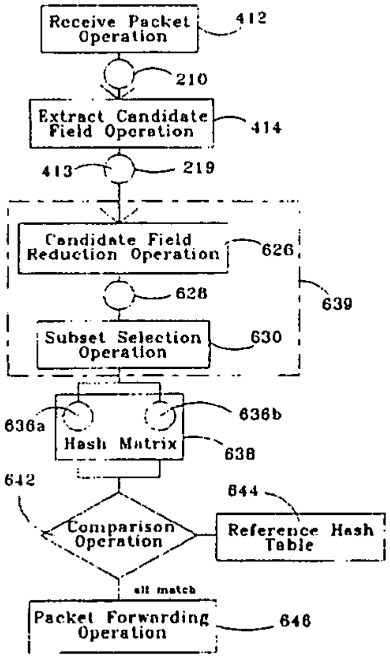
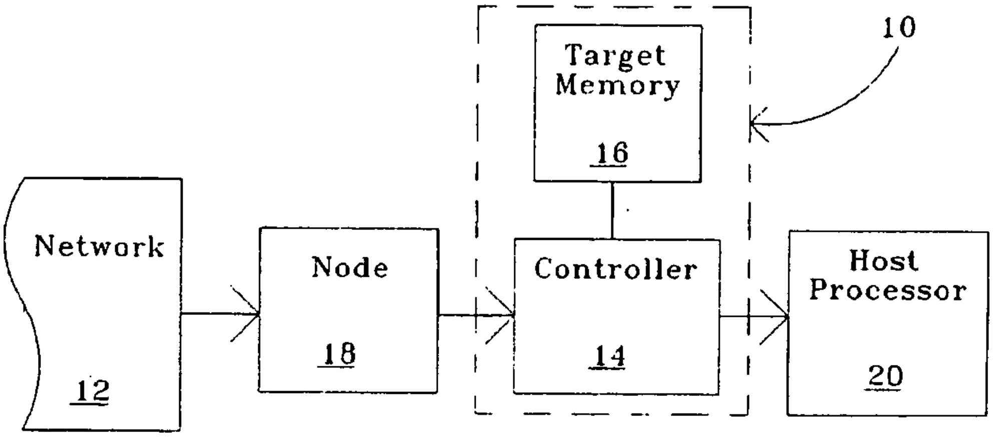
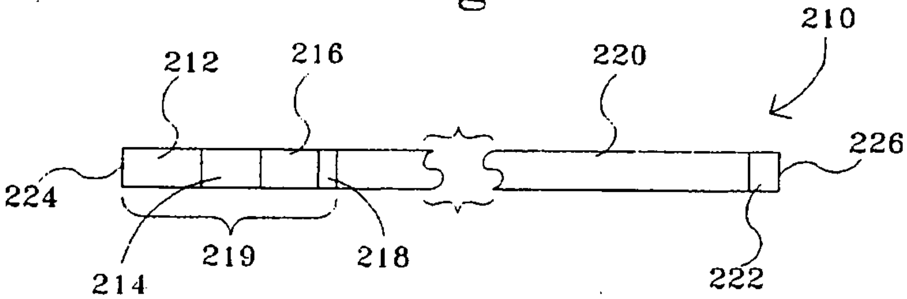


Fig. 1



10
Target Memory
16
Network
12
Node
18
Controller
14
Host Processor
20

Fig. 2



210
212
216
220
226
224
219
218
214
222

Fig. 3

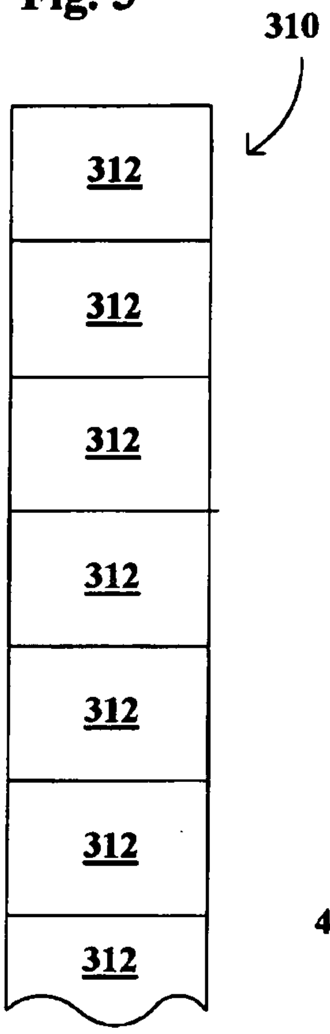


Fig. 4 (prior art)

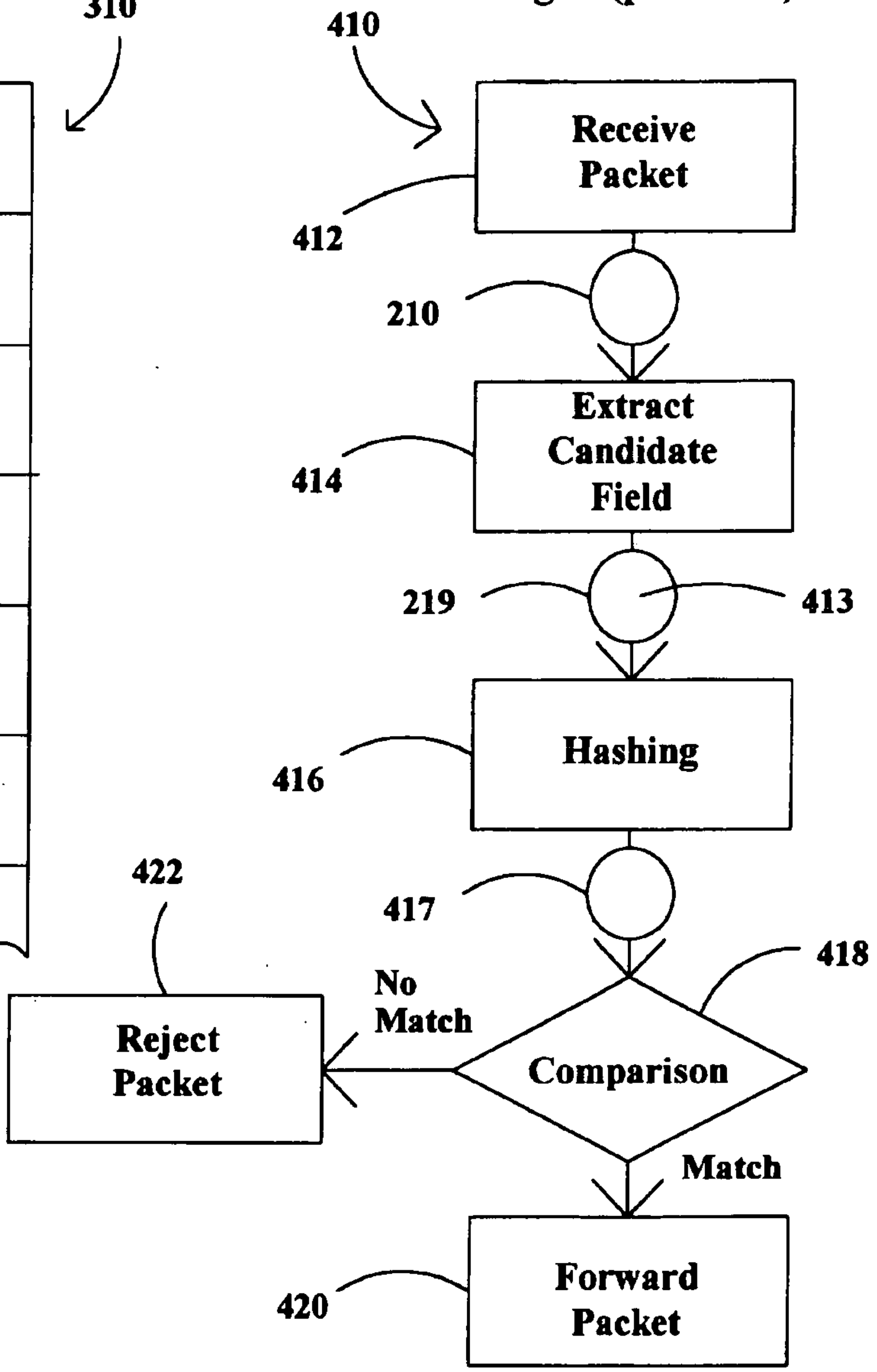


Fig. 5

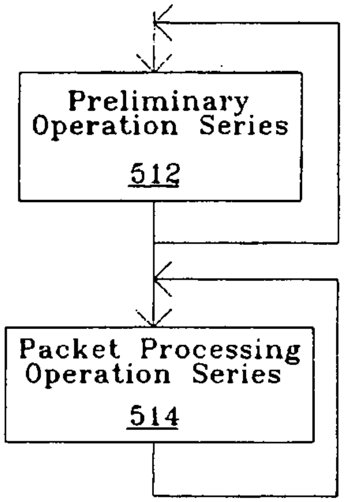

Preliminary Operation Series
512
Packet Processing Operation Series
514

Fig. 6

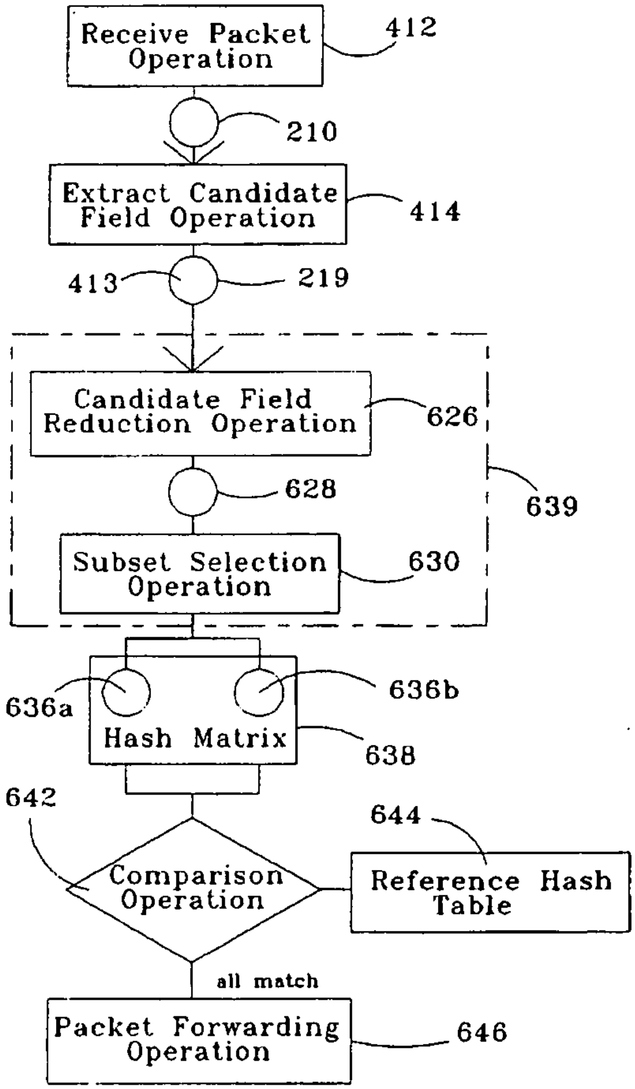
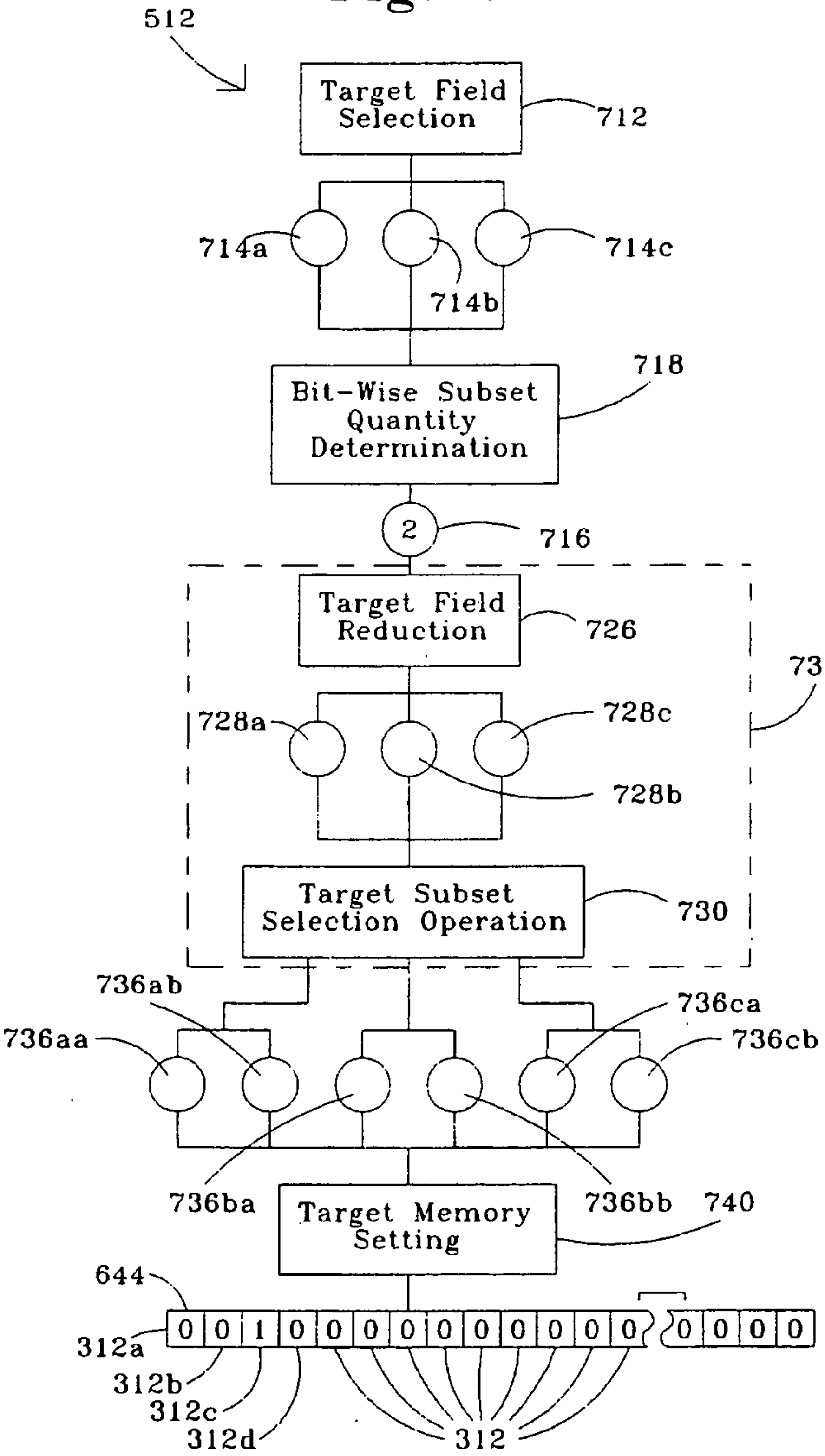

Receive Packet Operation
412
210
Extract Candidate Field Operation
414
413
219
Candidate Field Reduction Operation
626
628
639
Subset Selection Operation
630
636a
636b
Hash Matrix
638
642
644
Comparison Operation
Reference Hash Table
all match
Packet Forwarding Operation
646

Fig. 7


512
Target Field Selection
712
714a
714b
714c
Bit-Wise Subset Quantity Determination
718
2
716
Target Field Reduction
726
739
728a
728c
728b
Target Subset Selection Operation
730
736ab
736ca
736aa
736cb
736ba
736bb
740
Target Memory Setting
644
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
312a
312b
312c
312d
312

PACKET FILTERING FOR DATA NETWORKS

TECHNICAL FIELD

The present invention relates generally to the field of computer science and more particularly to data networking and component devices attached to data networks.

BACKGROUND ART

Computer networks are becoming increasingly common in industry, education and the public sector. The media over which data are carried generally carry data in units referred to as "packets" which are destined for many different sources. Addressing and packet typing are included in most standardized and proprietary packet based networking protocols which make use of destination address fields at the beginning of and/or within each data packet for the purpose of distinguishing proper recipient(s) of the data of the packets. As a packet is received at intermediate and end components in a system, rapid determination of the proper recipient (s) for the data must be made in order to efficiently accept, forward, or discard the data packet. Such determinations are made based upon the above discussed address, packet type and/or other fields within the relevant packets. These determinations can be made by network controller hardware alone, by a combination of hardware and software, or by software alone. In broadcast type networks, every node is responsible for examining every packet and accepting those "of interest", while rejecting all others. This is called "packet filtering". Accuracy, speed and economy of the filtering mechanism are all of importance.

When the above discussed determinations are made through a combination of hardware and software, the hardware is said to have accomplished a "partial filtering" of the incoming packet stream. It should be noted that one type of packet filtering is accomplished on the basis of packet error characteristics such as collision fragments known as "runts", frame check sequence errors, and the like. The type of filtering relevant to the present discussion is based upon packet filtering in which filtering criteria can be expressed as simple Boolean functions of data fields within the packet as opposed to filtering based upon detection of errors or improperly formed packets.

In the simplest case, each node of a computer network must capture those packets whose destination address field matches the node's unique address. However there frequently occur situations in which additional packets are also of interest. One example occurs when the node belongs to a predefined set of nodes all of which simultaneously receive certain specific "groupcast" packets which are addressed to that group. Groupcast packets are usually identified by some variation of the address field of the packet. Groupcast address types generally fall into one of two forms. "Broadcast" addresses are intended for all nodes and "multicast" addresses are targeted for specific applications to which subsets of nodes are registered. Another case of such field-based packet filtering occurs when certain network management nodes are adapted to focus on specific protocols, inter-node transactions, or the like, to the exclusion of all other traffic.

Attachment of a networked device to the network is realized through a "controller" which operates independently of the host processor. Packet filtering then occurs in two successive stages beginning at the controller, which examines packets in real-time. To accomplish this, the controller is "conditioned" with an appropriate subset of the specified filtering criteria, according to the filtering capabilities of that controller. The controller classifies packets into three categories: Those not satisfying the filter criteria ("rejects"); those satisfying the criteria ("exact matches"); and those possibly satisfying the criteria ("partial matches"). Rejects are not delivered to the processor. Those packets which are classified as exact or as possible matches are delivered, with appropriate indications of their classification, to the device processor. The controller, ideally, excludes as many unwanted packets as its capabilities will allow, and the host processor (with the appropriate software operating therein) completes the overall filtering operation, as required. The value of filtering packets at the controller level (the partial filtering) is that it reduces the burden on the host processor.

Controller filtering implementations are constrained by the fact that they must process packets in real-time with packet reception. This places a high value on filtering mechanisms that can be implemented with a minimum amount of logic and memory. Controller based filtering criteria are contained in a target memory. In the case of exact matching, a literal list of desired targets is stored in the target memory. While exact matching provides essentially perfect filtering, it can be used in applications wherein there are only a very small number of targets.

Partial filtering is employed when the potential number of targets is relatively large, such as is often the case in multicast applications. A primary consideration is the "efficiency" of the partial filter. Efficiency (E), in this context, may be expressed as:

$$E=Tn/Pn$$

where:

Tn=the number of target packets of interest; and

Pn=the number of potential candidates delivered to the processor.

An efficiency of E=1.0 represents an exact filtering efficiency wherein every candidate is a desired target. This is the efficiency of the filtering which occurs in the "exact matching" previously discussed herein. While exact filtering efficiency is an objective, the previously mentioned constraints, including that the controller must do its filtering in essentially real-time, will generally not allow for such efficiency.

The predominant method used in the prior art for partial packet filtering is "hashing". The process conventionally begins with the extraction from each received packet of all fields involved in the specified filtering criteria. The composite of such relevant fields is called the "candidate field". Assuming an even distribution of candidate fields (a situation that is not always literally accurate, but the assumption of which is useful for purposes of analysis), there will be a potential number of packet candidates of 2^{Cb} where Cb is the number of bits in the candidate field. The hashing function produces a reduction in the bit size of the candidate field according to a "hashing function". As a part of the initiation of the controller, the hashing function is applied to each field of the target memory to assign a "target hash value" to each such field. The controller memory is initialized as a bit mask representing the set of target hash values. Then, during operation, a "candidate hash value" is created by applying the hashing function to each candidate field. The candidate hash value is used as a bit index into the controller memory, with a match indicating a possible candidate.

As can be appreciated in light of the above discussion and from a general understanding of simple hashing operations, the hashing function has the effect of partitioning the 2^{Cb} candidate possibilities into Mb groups (called "buckets"), where Mb is the number of bits in the controller's target memory. Because candidate packets that fall into the same bucket are not distinguished, a "hit" represents any of 2^{Cb}/Mb candidates. Useful hashing functions will partition the candidate possibilities in a roughly uniform distribution across the set of Mb buckets. For a single target, the efficiency of such a hashing method is Mb/2^{Cb} If Tn desired targets are represented by Bn buckets (where Bn<=Tn and Bn<=Mb, the efficiency of such a hashing method is:

$$E=Tn/(Bn2^{Cb}/Mb)=TnMb/Bn2^{Cb}$$

In exact matching, target memory could hold Mb/Cb targets. Hashing is appropriate when the number of buckets (Bn) is larger than this figure. However, effective hashing also requires that the number of buckets be less than Mb, because as target memory density increases there is less differentiation among candidate fields. With the target memory full of hash targets, Bn=Mb and the efficiency is Tn/2^{Cb}.

As can be appreciated, the described prior art hashing method used for partial packet filtering implies a loss of information in that a single hash value potentially represents a large set of candidates. Clearly, it would be desirable to reduce such loss of data. Correspondingly, it would desirable to maximize the filtering efficiency for a given Mb or (or to minimize the Mb for a given filter efficiency).

To the inventor's knowledge, no prior art method for partial packet filtering has improved efficiency or reduced data loss as compared to the conventional hashing method described above.

DISCLOSURE OF INVENTION

Accordingly, it is an object of the present invention to provide a method and means for efficiently performing a partial filtering operation on data packets in a computer network.

It is another object of the present invention to provide a method and means for partial packet filtering which rejects a maximum number of incoming packets which are not at interest without requiring a large target memory and without unduly slowing down the processing of incoming packets.

It is still another object of the present invention to provide a partial packet filtering method and means which is inexpensive to implement.

It is yet another object of the present invention to provide a partial packet filtering method and means which will operate in real-time or near real-time.

It is still another object of the present invention to provide a partial packet filtering method and means which is adaptable to a variety of network system requirements.

Briefly, the preferred embodiment of the present invention implements multiple independent hashing functions applied in parallel to the candidate field of each packet. The combined application of multiple independent hashing functions results in specification of a hash matrix, with each coordinate of the hash matrix being the result of one of the hashing functions. The hash matrix includes the results of different hashing algorithms applied to a single candidate field, or the same hashing function applied to different subsets of the candidate field, or a combination thereof. The filter parameters consist of the set of acceptable result values for each hashing operation.

An advantage of the present invention is that partial packet filtering efficiency is improved, thereby freeing the host processor from a substantial portion of the packet filtering operation.

Yet another advantage of the present invention is that filtering efficiency is increased geometrically with an increase in target memory.

Still another advantage of the present invention is that a minimum amount of target memory is required for a specific target efficiency.

Yet another advantage of the present invention is that the partial packet filtering can be performed in a minimum amount of time for a given target efficiency.

These and other objects and advantages of the present invention will become clear to those skilled in the art in view of the description of the best presently known modes of carrying out the invention and the industrial applicability of the preferred embodiments as described herein and as illustrated in the several figures of the drawing.

BRIEF DESCRIPTION OF THE DRAWING

FIG. 1 is a block diagram depicting a portion of a computer network with an improved partial packet filter according to the present invention in place therein;

FIG. 2 is a diagrammatic representation of a conventional prior art Ethernet data packet;

FIG. 3 is diagrammatic representation of a hash table;

FIG. 4 is a flow chart showing a conventional prior art partial packet filtering operation;

FIG. 5 is a block depiction of a partial packet filtering method according to the present invention;

FIG. 6 is a flow chart, similar to the chart of FIG. 4, depicting the packet processing operation series of FIG. 5; and

FIG. 7 is a flow chart depicting the preliminary operation series of FIG. 5.

BEST MODE FOR CARRYING OUT INVENTION

The best presently known mode for carrying out the invention is a partial packet filter for implementation in a personal computer resident Ethernet controller. The predominant expected usage of the inventive improved packet filter is in the interconnection of computer devices, particularly in network environments where there are relatively few targets.

The improved partial packet filter of the presently preferred embodiment of the present invention is illustrated in a block diagram in FIG. 1 and is designated therein by the reference character **10**. In the diagram of FIG. **1**, the improved partial packet filter **10** is shown configured as part of a network system **12** (only a portion of which is shown in the view of FIG. 1). In many respects, the best presently known embodiment **10** of the present invention is structurally not unlike conventional partial packet filter mechanisms. Like prior art conventional partial packet filters, the best presently known embodiment **10** of the present invention has a controller **14** with an associated target memory **16**. In the example of FIG. 1, the improved partial packet filter **10** receives data from a network node **18** and performs the inventive improved packet filtering process on such data before passing selected portions of the data on to a host processor 18 to which the improved partial packet filter 10 is dedicated.

FIG. 2 is a diagrammatic representation of a conventional Ethernet data packet 210. The standardized Ethernet packet 210 has a preamble 212 which is 64 bits in length, a destination address 214 which is 48 bits in length, a source address 216 which is 48 bits in length, a length/type field 218 which is 16 bits in length and a data field 220 which is variable in length from a minimum of 46 eight bit bytes to a maximum of 1500 bytes. Following the data field 220 in the packet 210 is a 4 byte (32 bit) frame sequence check ("FCS") 222. The packet 210 is transmitted serially beginning at a "head" 224 and ending at a "tail" 226 thereof. The preamble 212, destination address 214, source address 216 and length/type field 218 are collectively referred to as the header 219.

FIG. 3 is a diagrammatic representation of a conventional single dimensional hash table 310 with which one skilled in the art will be familiar. The hash table 310 has a plurality of address locations 312 each of which can be "set" (set to 1) or left unset (set to zero).

FIG. 4 is a flow diagram depicting the operation of a conventional prior art partial packet filtering operation 410. As previously discussed briefly, a packet 210 (FIG. 2) is received (receive packet operation 412) from the network 18 (FIG. 1) and a candidate field 413 (such as the header 219 of the packet 210) is extracted (extract candidate field operation 414). A hashing operation 416 is performed on the extracted candidate field 413 to produce a hash value 417 and the hash value 417 is compared to the hash table 310 (FIG. 3) stored in the target memory 16 (FIG. 1) in a comparison operation 418. If the result of the comparison operation 418 is a match, the packet 210 is forwarded in a forward packet operation 420. If the result of the comparison operation 418 is not a match, the packet 210 is rejected 422 in a reject packet operation. It should be remembered that the use of the header 219 here is an example only, and any portion or combined portions of the packet 210 might constitute the candidate field 413 in a given application.

FIG. 5 is a flow diagram depicting the inventive improved packet filtering process 510. The improved packet filtering process 510 is accomplished in a preliminary operation series 512 and a packet processing operation 514, each of which is repeated as required, as will be discussed hereinafter. The preliminary operation series 512 is accomplished according to software residing in the host processor 20 (FIG. 1) to configure the target memory 16 (FIG. 1) as will be discussed hereinafter. It should be noted that the fact that the improved packet filtering process 510 is divided into the two main operation categories (the preliminary operation series 512 and the packet processing operation 514) does not distinguish this invention over the prior art. Rather, the processes within the preliminary operation series 512 and the packet processing operation 514 describe the essence of the inventive process.

FIG. 6 is a flow chart showing the inventive packet processing operation 514 in a manner analogous to the presentation of the prior art partial packet filtering operation 410 depicted in FIG. 4. As can be seen in the view of FIG. 6, the packet processing operation series 514 is similar in many respects to the prior art partial packet filtering process 410 (FIG. 4). In the packet processing operation series 514, a packet 210 (FIG. 2) is received (receive packet operation 412) and a candidate field 413 is extracted in an extract candidate field operation 414. In the best presently known embodiment 10 of the present invention, the inventive packet processing operation series 514 next performs a candidate field reduction operation 626. In the best presently known embodiment 10 of the present invention, the candidate field reduction operation 626 is merely the application of the conventional CRC polynomial algorithm to the candidate field 413 to yield a 32 bit CRC output value 628 (although any of a number of similar algorithms might be applied for this purpose). Next, a subset selection operation 630 selects a predetermined number (two in the example of FIG. 6) of bit-wise subsets 636 from the CRC output value 628. The method for determining the quantity of bit-wise subsets 636 to be selected in the subset selection operation 630, and the size of each, will be discussed hereinafter. In the best presently known embodiment 10 of the present invention, the bit-wise subsets 636 are each 6 bits in length. It should be noted that, in the best presently known embodiment 10 of the present invention, the bit-wise subsets 636 are selected from the CRC output value 628 simply by taking the first 6 bits of the CRC output value 628, the second six bits, and so on until as many bit-wise subsets as are needed are obtained and so, in the best presently known embodiment 10 of the present invention, the bit wise subset 636 are "consecutive bit section " of the fixed size field (the CRC output value 628 in the best presently known embodiment 10 of the present invention. The inventors have determined that the bits of the CRC output value 628 (resulting from the CRC polynomial function) are independent of each other, and so any 6 bit portion of the CRC output value 628 is as representative of the CRC output value 628 as is any other 6 bit portion.

The bit-wise subsets 636 are then compared to the hash table 310 (FIG. 3) stored in the target memory 16 (FIG. 1) in a comparison operation 642. The combined multiple hash values 636 may be considered to be a hash matrix 638 (in the example of FIG. 6, a two dimensional hash matrix 638).

It is important to note that the essence of the present inventive method lies in the extraction of the plurality of independent or relatively independent representative indices of the candidate field 413 ("candidate filled indices") which, in the example of the best presently known embodiment 10 of the present invention are the bit-wise subsets 636 which make up the hash matrix 638. That is, the bit-wise subsets 636 are representative fields in that the bit-wise subsets 636 are representative of the candidate field 413, as discussed above. The generally simultaneous (parallel) processing of these is the source of the advantages of the present inventive method and means. The exact method described herein in relation to the best presently known embodiment 10 of the present invention, that of first reducing the candidate field 413 in the candidate field reduction operation 626 and then extracting the bit-wise subsets 636 is but one of many potential methods for accomplishing such a parallel hashing operation 639, and the present invention is not intended to be limited by this aspect of the best presently known embodiment 10.

In the best presently known embodiment 10 of the present invention, in a comparison operation 642, each of the bit-wise subsets 636 is compared to a reference hash table 644 (a "target hash array") stored in the target memory 16 (FIG. 1) and only if all match is the packet 210 forwarded in a packet forwarding operation 646. In the example of FIG. 6, the reference hash table 644 will be a 64 element array representing all values from 0 through 63 inclusive. Some elements of the reference hash table 644 are set as will be discussed hereinafter in relation to the preliminary operation series 512. If the value of the bit-wise subset "falls into one of the buckets" (is equivalent to a corresponding set bit in the reference hash table **644**), then the data packet **210** is defined as being a "match".

Now returning to a consideration of the preliminary operation series **512** (FIG. **5**) with an understanding of the packet processing operation series **514**, the target memory **16** is configured in process steps much like those described in relation to the packet processing operation series **514** of FIG. **6**.

FIG. **7** is a flow diagram of the preliminary operation series **512** according to the best presently known embodiment **10** of the present invention. A preliminary operation which is common to both the prior art and the present invention is a target field(s) selection process **712**. The target (field) s selection process is merely the selection of criteria to which incoming packets **210** are to be compared. For example, if the entire process is to be on the basis of desired destinations, then an intended destination address **214** (FIG. **2**) will be (one of) the target field(s) **714**, and if three destinations are of interest, then there will be three target fields **714** as illustrated in the example of FIG. **7**. The actual process involved in selecting the target field(s) is a function of network control software which is found in the prior art and which is not relevant to the present invention except to the extent that it delivers the target field(s) **714** to the inventive preliminary operation series **512**.

Having determined the quantity of target fields **714** of interest, host software will next determine a bit-wise subset quantity **716** (the appropriate "subset quantity" of bit-wise subset **636**) in a bit-wise subset quantity determination operation **718**. The bit-wise subset quantity determination operation **718** will be discussed in more detail hereinafter, as it can be better understood in light of the present description of the entire preliminary operation series **512**. For the present simplified example of FIGS. **6** and **7**, and as already mentioned, the bit-wise subset quantity **716** is two. That is, two of the bit-wise subsets **636** are to be extracted from the CRC output value **628** in the subset selection operation **630** of FIG. **6**.

As can be appreciated, the target fields **714** are each equivalent in form to the candidate fields **413** discussed previously herein, and processing of the target fields **714** is much the same as has been previously described herein in relation to the candidate fields **413**. In the inventive preliminary operation series **512**, each of the target fields **714** is processed in a target field reduction operation **726** by application of the CRC polynomial to produce a target CRC value **728**. Each of the target CRC values **728** is then processed in a target subset selection operation **730** to produce a plurality (two for each target CRC value **728** for a total of six, in the present example) of target bit-wise subsets **736**. In more general terms, each of the "target fields **714** (having been selected according to prior art methods as discussed previously, herein) is processed as described to produce a "target representative field" (the target CRC value **728** in the present example), which is then further processed as described to produce the "target indices", which target indices may be "target string subsets**38** of the target representative field and which are, in the present example, the target bit-wise subsets **736**. This process is alike to the process which is repeated as necessary to process each incoming data packet **210**, wherein the candidate fields **413** are processed to produce a candidate representative field (the CRC output value **628** in the present example), which is further processed to produce the "candidate string subsets" (the bit-wise subsets **636** in the present example). The quantity of target bit-wise subsets **736** taken from each target CRC value **728** is also the bit-wise subset quantity **716** (two, in the present example). It should be noted that a target parallel hashing operation **739** is like the previously described parallel hashing operation **639** in that the invention might be practiced with variations of the specific steps therein which are presented here as features of the best presently known embodiment **10** of the present invention.

In a target memory setting operation **740** the reference hash table **644** is formatted such that each memory location **312** corresponding to a value of any of the target bit-wise subsets **736** is set. For example, if the first target bit-wise subset **736***a* were "000010" (decimal value 2) then the third memory location **312***c* in the reference hash table **644** would be set to "1", as is illustrated in FIG. **7**. As can be appreciated from the above discussion, the maximum number of memory locations **312** in the reference hash table **644** which can be set by this process is the quantity of target bit-wise subsets **736** (six, in the present example). However, since two or more of the target bit-wise subsets might coincidentally hash to the same value, a lesser quantity of memory locations **312** might also be set.

Now returning to a more detailed discussion of the bit-wise subset quantity determination operation **718**, the target memory **16** is to be configured to maximize the effectiveness of the filtering based on the quantity of multicast packets **210** of interest to the software of the host processor Therefore, the bit-wise subset quantity determination operation **718** attempts to determine (or, at least, to approximate) an optimal number of indices per packet (and, thus, the bit-wise subset quantity **716** discussed previously herein). The "optimal" number here means that which will minimize the number of "uninteresting" packets which match the set data bits **312** in the reference hash table **644** while matching all of the "interesting" packets **210**. In the best presently known embodiment **10** of the present invention, the following table is used to determine the bit-wise subset quantity **716**.

TABLE OF SUBSET QUANTITIES

Addresses of Interest	Number of Hash Indices Bit-Wise Subset Quantity 716
1–2	5
3	4
4–9	3
10–16	2
17 or more	1

The above table is offered here as a guide only, in that the "optimal" number of selected hash indices may vary in ways not presently contemplated. Furthermore, it should be noted that the above table is based upon an assumption that none of the target indices (the target bitwise subsets **736** in the best presently known embodiment **10** of the present invention hash to the same memory locations **312** in the reference hash table **644**. If, indeed, two or more of the target bit-wise subsets **736** did hash to the same memory location **312**, then additional hash indices could be added to increase efficiency without sacrificing speed or requiring additional memory or processing.

It should be noted that while the packet processing operation series **514** is accomplished in the hardware of the best presently known embodiment **10** of the present invention, the preliminary operation series (which can be accomplished at a more leisurely pace) is performed primarily by software of the host processor **20**. As can be appreciated in light of the above discussion, the preliminary operation series will be repeated when the network **12** is reconfigured, when it is desired to communicate with additional members of the network **12**, or upon other occasions according to the needs of the user and the network **12**. The packet processing operation series **514** will be repeated whenever an incoming packet is detected from the network node **18**.

It should also be noted that, while the best presently known embodiment **10** of the present invention hashes each of the CRC values **628** and **728** to a common reference hash table **644**, the invention might be practiced with equal efficiency by hashing each of the CRC values **628** and **728** to its own individual hash table (not shown). Using the quantities of the example of FIGS. **6** and **7**, each of the individual hash tables would be 32 bits (memory locations **312**) large (one half of **64** bits, since it must be divided between the two target CRC values **728**). The individual bit-wise subsets **636** and **736** would then be 5 bits long (decimal value 0 through 31).

Various modifications may be made to the inventive improved packet filter **10** without altering its value or scope. For example, the quantity, size, and derivation of the plurality of bit-wise subsets **636** and **738** could readily be revised according to the parameters discussed herein.

All of the above are only some of the examples of available embodiments of the present invention. Those skilled in the art will readily observe that numerous other modifications and alterations may be made without departing from the spirit and scope of the invention. Accordingly, the above disclosure is not intended as limiting and the appended claims are to be interpreted as encompassing the entire scope of the invention.

INDUSTRIAL APPLICABILITY

The improved partial packet filter **10** is adapted to be widely used in computer network communications. The predominant current usages are for the interconnection of computers and computer peripheral devices within networks and for the interconnection of several computer networks.

The improved partial packet filters **10** of the present invention may be utilized in any application wherein conventional computer interconnection devices are used. A significant area of improvement is in the inclusion of the parallel processing of a plurality of indices (bit-wise subsets **636**) of a packet

The efficiency of the filtering provided by the improved partial packet filter **10** is significantly improved, particularly for cases where the number of targets is small relative to the number of "buckets" (memory locations **312**). To compare the efficiency of the present inventive improved packet filtering process **510** embodied in the improved partial packet filter **10** with the prior art partial packet filtering process **410**, assume, for example, the following values:

Mb=64 (representing 64 memory locations **312** in the reference hash table **644**)

Cb=48 (representing a 48 bit candidate field **413** size—a typical size of the destination address **214**

Dn=4 (representing a bit-wise subset quantity **716** of four)

Then, the prior art partial packet filtering process **410** will partition the 2^{Cb} possibilities among 64 distinct buckets, one of which matches the bucket into which the single target falls. In the improved packet filtering process **510**, the four parallel hashing functions partition among 16 possible buckets each. The efficiency (Ef) for the prior art partial packet filtering process **410** would then be:

$$Ef=64/2^{48}=\frac{1}{2}^{42}$$

The efficiency (Ef4) for this example of the improved packet filtering process **510** is:

$$Ef4=64^{4}/(4^{4}\cdot 2^{48})=\frac{1}{2}^{32}$$

The efficiency Ef4 is better than the efficiency Ef by a factor of 2^{10} (1024), which is to say that only a thousandth as many (uninteresting) packets will be delivered to the next stage of filtering using the inventive improved partial packet filter **10** as compared to the prior art.

Filtering of packets may be accomplished through a combination of exact and partial match filters. Typically, one or more partial filterings will occur first, with the multiple dimensions of each filtering accomplished in parallel with each other (according to the present invention). Packets which pass through the inventive improved partial packet filter **10** may then be filtered using an exact match filter technique, such as "binary search lookup" of the filter data in a sorted table of acceptable filter data values. Furthermore, results of partial filtering can be used to determine which of many (possibly sorted) tables in which to search for the packet.

Accordingly, the inventive improved packet filtering process **510** may be applied more than once to each incoming packet **210** (in a first stage and a second stage). In such an example, configuration of the first stage partial filtering would involve specification of the number and type of hashing operations to be performed, along with the portion of the packet which is to comprise the filter data for each such operation, along with acceptable results for each. Multiple partial filterings may be configured with the specification including the logical relation to apply to the results of each filtering. For example, partial filtering A might be to apply the 32 bit CRC polynomial to the destination address field of an Ethernet packet, and retain the lowest order 3 bits—a value from 0 to 7. Partial filtering B might be to apply the 32 bit CRC polynomial to the source address field of the Ethernet packet, and retain the lowest order 3 bits. The logical relation might be to accept packets only for which the results of the first filtering (A) is either 2 or 4, and the result of the second filtering (B) is either a 3 or a 4. In a general case, one may expect the likelihood of arbitrarily filter data to "pass" the first filtering to be 2 in 8 (25%), since 2 of the 8 values from 0 to 7 are acceptable. Similarly, the likelihood of the second filtering "passing" such a filter is 2 in 8 (25%). Assuming that the two filterings are, as desired, truly independent, the likelihood of this arbitrary packet being accepted is the product of these, or 1 in 16. Note further that the specification of these "acceptable result sets" ({2,4} for A and {3,4} for B) requires 16 bits of information for full specification, where 8 bits indicate the acceptability/unacceptability of each of the 8 possible values of filtering A, and 8 additional bits indicate the acceptability/unacceptability of each of the 8 possible values of filtering B. Use of such multiple partial filterings may be especially effective in situations where filtering criteria are derived from independent portions of the filter data, such as filtering for all packets whose destination address OR whose source address is within a set of interesting addresses AND whose packet type indicates a particular protocol of interest.

Since the improved partial packet filters of the present invention may be readily constructed and are compatible with existing computer equipment it is expected that they will be acceptable in the industry as substitutes for conventional means and methods presently employed for partial packet filtering. For these and other reasons, it is expected that the utility and industrial applicability of the invention will be both significant in scope and long-lasting in duration.

We claim:

1. A method for selectively forwarding a data packet and controlling the distribution of data packets in a computer network system, the data packet having a candidate field containing information about the data packet, the method comprising:

configuring a target memory of a controller to contain a target hash array in steps including;

aa determining a target field and extracting a plurality of target indices from said target field, the target indices being a binary number having a value;

ab setting memory locations in the target memory corresponding to the value of each of the target indices; and

processing the data packet in steps including:

ba extracting the candidate field from the data packet;

bb extracting from the candidate field a plurality of candidate field indices;

bc comparing the values of each of the candidate field indices to the target hash array; and

bd forwarding the packet when each of the values of each of the candidate field indices corresponds to a memory location of the target hash array which was set in step ab.

2. The method of claim 1, wherein:

step aa is accomplished in substeps including:

aa1 reducing the target fields to a plurality of target representative fields; and

aa2 selecting one or more target string subsets from the target representative field; and

step bb is accomplished in substeps including:

bb1 reducing the candidate field to a plurality of candidate representative fields; and

bb2 selecting one or more candidate string subsets from the target representative field.

3. The method of claim 1, wherein:

step ab is accomplished by causing only those memory locations in the target memory which correspond to the value of each of the target string subsets to contain a value of one.

4. The method of claim 2, wherein:

step aa1 is accomplished by applying a cyclic redundancy check algorithm to each of the target fields; and

step bb1 is accomplished by applying the same cyclic redundancy check algorithm to the candidate field.

5. The method of claim 2, wherein:

in step aa2 the target string subsets are selected by extracting a plurality of target bit-wise subsets from the target representative field; and

in step bb2 the candidate string subsets are selected by extracting a plurality of candidate bit-wise subsets from the representative candidate field.

6. The method of claim 2, and further including:

an additional process step preceding step ab wherein a subset quantity is determined, the subset quantity being the number of target string subsets to be extracted from each of the target representative fields and also the number of candidate string subsets to be extracted from each of the candidate representative fields.

7. The method of claim 6, wherein:

the additional process step is accomplished, at least initially, by selecting the subset quantity from a table of subset quantities.

8. The method of claim 2, wherein:

each of the target representative target and the candidate representative field are 32 bits in length.

9. The method of claim 1, wherein:

steps aa through ab are repeated when a change in the distribution of data packets is desired.

10. The method of claim 1, wherein:

steps ba through bd are repeated for each incoming data packet.

11. The method of claim 1, and further including:

an additional process step preceding step ab wherein a subset quantity is determined, the subset quantity being the number of target indices to be extracted from each of the target fields and also the number of candidate indices to be extracted from each of the candidate fields.

12. The method of claim 11, wherein:

the additional process step is accomplished, at least initially, by selecting the subset quantity from a table of subset quantities appropriate to a quantity of target quantities.

13. The method of claim 1, wherein:

the candidate field includes a target address field of the data packet.

14. The method of claim 1, wherein:

the data packet is a standardized Ethernet data packet.

15. The method of claim 1, wherein:

the target hash array is an unapportioned array such that each of the target indices is used to set memory locations in that unapportioned array.

16. The method of claim 1, wherein:

the target hash array is apportioned such that at least some of the target indices are directed to different portions of the target hash array.

17. The method of claim 1, wherein:

the target indices and the candidate indices are each a binary string of fixed bit length.

* * * * *

US006266700B1

(12) **United States Patent**
Baker et al.

(10) **Patent No.: US 6,266,700 B1**
(45) **Date of Patent: *Jul. 24, 2001**

(54) **NETWORK FILTERING SYSTEM**

(76) Inventors: **Peter D. Baker**, 36 Blackbird La., Aliso Viejo, CA (US) 92656; **Karen Neal**, 1326 Saltair Ave., #6, Los Angeles, CA (US) 90025

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **09/113,704**

(22) Filed: **Jul. 10, 1998**

Related U.S. Application Data

(63) Continuation of application No. 08/888,875, filed on Jul. 7, 1997, now Pat. No. 5,781,729, which is a continuation of application No. 08/575,506, filed on Dec. 20, 1995, now Pat. No. 5,793,954.

(51) **Int. Cl.**^{7} **G06F 15/16**; H04L 12/28

(52) **U.S. Cl.** **709/230**; 709/220; 709/223; 709/224; 709/250; 370/401

(58) **Field of Search** 709/200, 202–203, 709/213–214, 223–224, 227–230, 249–250, 220; 370/400–401; 711/100, 147–148, 170

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,121,342	*	6/1992	Szymborski et al.	709/224
5,515,513	*	5/1996	Metzger et al.	370/401
5,727,149	*	3/1998	Hirata et al.	709/250
5,764,899	*	6/1998	Eggleston et al.	709/203
5,781,729	*	7/1998	Baker et al.	709/230
5,793,954	*	8/1998	Baker et al.	709/250
5,796,954	*	8/1998	Hanif et al.	709/230
5,826,030	*	10/1998	Hebert	709/220

* cited by examiner

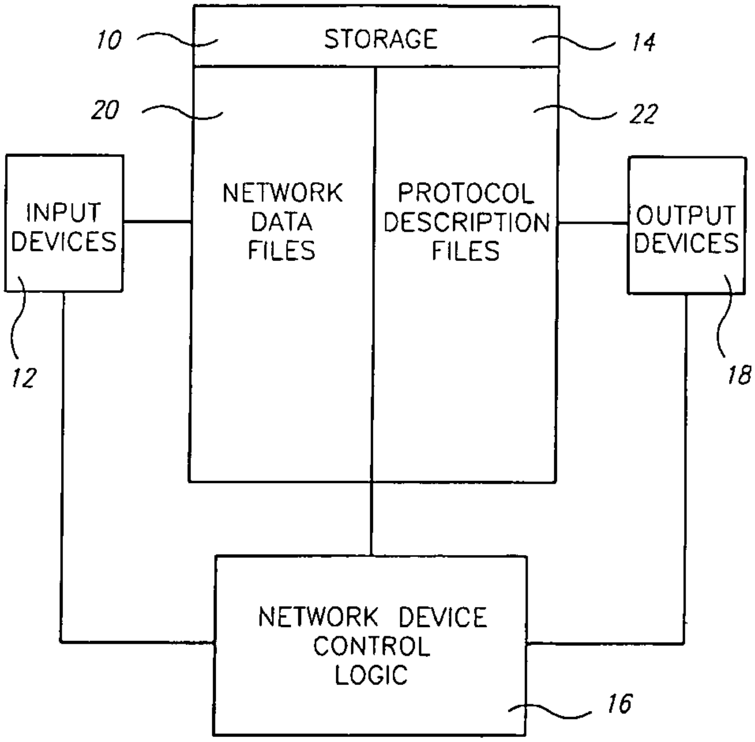
Primary Examiner—Bharat Barot
(74) *Attorney, Agent, or Firm*—Lyon & Lyon LLP

(57) **ABSTRACT**

A network interface system and related methods. A single logic control module, which may be implemented in hardware or software, is utilized to perform any of a number of data manipulation functions including, for example, parsing, filtering, data generation or analysis, based upon one or more programmably configurable protocol descriptions which may be stored in and retrieved from an associated memory.

1 Claim, 20 Drawing Sheets

Microfiche Appendix Included
(1 Microfiche, 84 Pages)



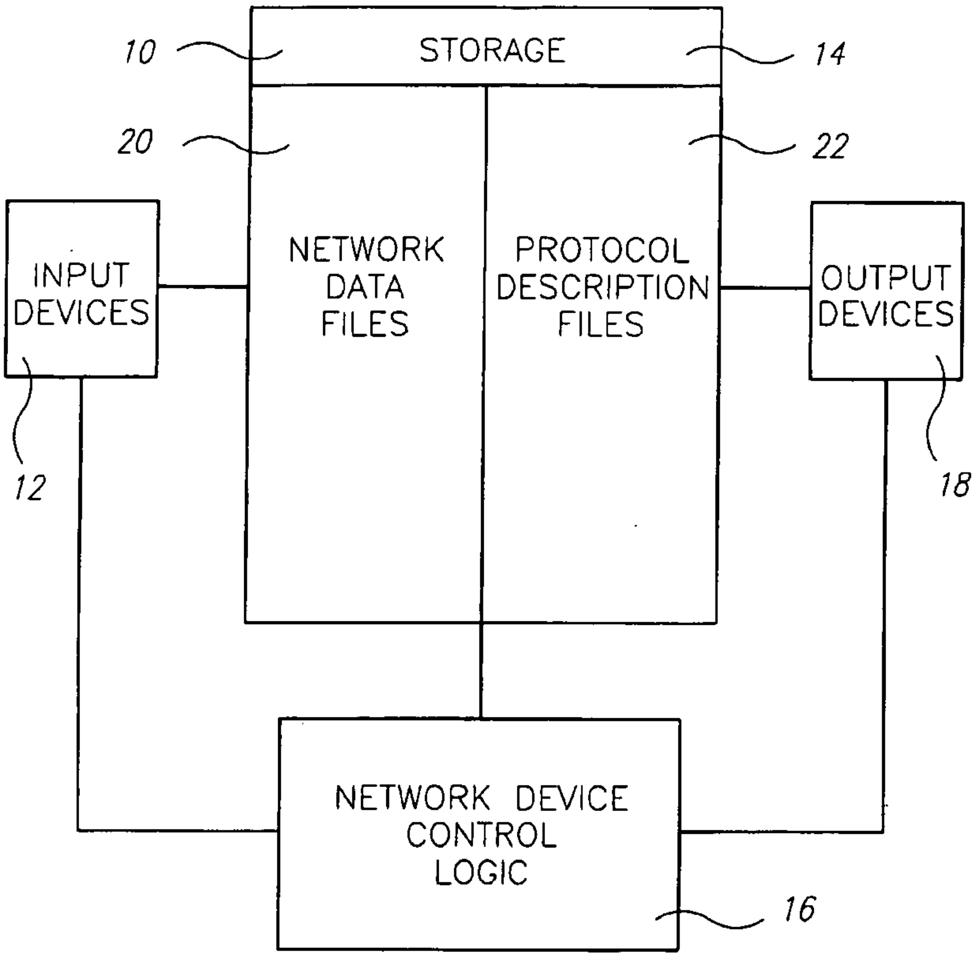

10
STORAGE
14
20
22
INPUT DEVICES
NETWORK DATA FILES
PROTOCOL DESCRIPTION FILES
OUTPUT DEVICES
12
18
NETWORK DEVICE CONTROL LOGIC
16

FIG. 1

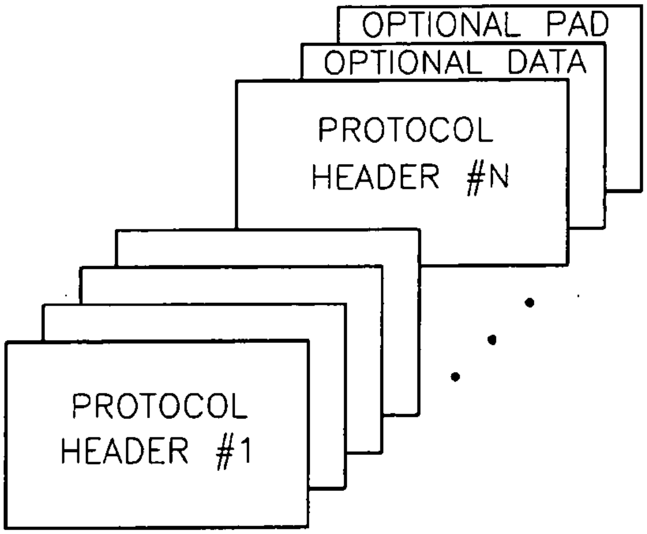

OPTIONAL PAD
OPTIONAL DATA
PROTOCOL HEADER #N
PROTOCOL HEADER #1

FIG. 2

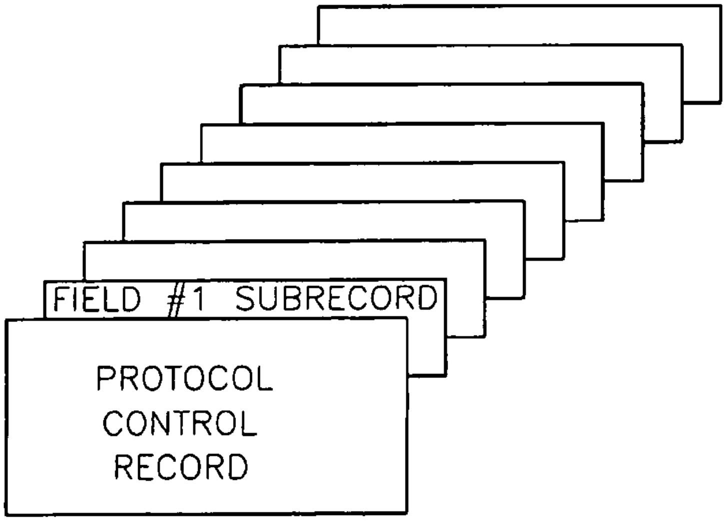

FIELD #1 SUBRECORD
PROTOCOL CONTROL RECORD

FIG. 3

ETHERNET CONTROL RECORD					
PROTOCOL NAME	NUMBITS	NUMFIELDS	CURFIELD	FIELDS	OPTIONS
ETHERNET MAC HEADER	112	5	0	FIGURE 4A	[NONE]

FIG. 4

INDEX	FIELD NAME	BIT OFFSET	BIT LENGTH	LEFT SHIFT	RIGHT SHIFT	CHECK SUM	FRAME LENGTH	HEADER LENGTH	STATISTICS	LOOKUP STRUCTURE	FILTER	FORMAT
0	DST VENDOR ADDRESS	0	24	0	16	0	0	0	[NONE]	FIG 4B	FIG 10, LDX 1	HEX
1	DST STATION ADDRESS	24	24	0	16	0	0	0	[NONE]	[NONE]	FIG 10, LDX 1	HEX
2	SRC VENDOR ADDRESS	48	24	0	16	0	0	0	[NONE]	FIG 4C	[NONE]	HEX
3	SRC STATION ADDRESS	72	24	0	16	0	0	0	[NONE]	[NONE]	[NONE]	HEX
4	TYPE	96	16	0	16	0	0	0	[NONE]	FIG 4D	[NONE]	HEX

FIG. 4A

DESTINATION VENDOR ADDRESS LOOKUP STRUCTURE					
PROTOCOL	NEXT INDEX	MINIMUM	MAXIMUM	MASK	TRANSLATION
[NONE]	1	0x000000	0x000000	ALL	"FAST ROUTERS, INC."
[NONE]	1	0x000001	0xFFFFFF	ALL	"UNKNOWN"

FIG. 4B

SOURCE VENDOR ADDRESS LOOKUP STRUCTURE					
PROTOCOL	NEXT INDEX	MINIMUM	MAXIMUM	MASK	TRANSLATION
[NONE]	3	0x000000	0x000000	ALL	"FAST ROUTERS, INC."
[NONE]	3	0x000001	0xFFFFFF	ALL	"UNKNOWN"

FIG. 4C

ETHERNET TYPE LOOKUP STRUCTURE					
PROTOCOL	NEXT INDEX	MINIMUM	MAXIMUM	MASK	TRANSLATION
[NONE]	5	0x0000	0x8887	ALL	"UNKNOWN"
FIGURE 5	5	0x8888	0x8888	ALL	"GP"
[NONE]	5	0x8889	0xFFFF	ALL	"UNKNOWN"

FIG. 4D

ETHERNET CONTROL RECORD					
PROTOCOL NAME	NUMBITS	NUMFIELDS	CURFIELD	FIELDS	OPTIONS
GP-GENERIC PROTOCOL	160	11	0	FIGURE 5A	FIGURE 6

FIG. 5

INDEX	FIELD NAME	BIT OFFSET	BIT LENGTH	LEFT SHIFT	RIGHT SHIFT	CHECK SUM	FRAME LENGTH	HEADER LENGTH	STATISTICS	LOOKUP STRUCTURE	FILTER	FORMAT
0	VERSION NO.	0	4	0	28	0	0	0	CNT/INDEX&CNT	[NONE]	[NONE]	DECIMAL
1	HEADERLEN	0	4	4	28	0	0	32	SUM/INDEX&CNT	FIG 5B	[NONE]	DECIMAL
2	FRAME LENGTH	0	16	8	16	0	8	0	SUM	[NONE]	[NONE]	DECIMAL
3	FRAME TYPE	0	8	24	24	0	0	0	INDEX&CNT	FIG 5C	FIG 10, LDX 2	HEX
4	CHECKSUM	4	16	0	16	PTR	0	0	[NONE]	[NONE]	[NONE]	HEX
5	CONTROL	4	8	16	24	0	0	0	[NONE]	[NONE]	[NONE]	BITFIELD
6	HOP COUNT	4	8	24	24	0	0	0	[NONE]	[NONE]	[NONE]	DECIMAL
7	SOURCE SOCKET	8	16	0	16	0	0	0	[NONE]	FIG 5D	[NONE]	HEX
8	DESTINATION SOCKET	8	16	16	16	0	0	0	[NONE]	FIG 5E	[NONE]	HEX
9	SOURCE ADDRESS	12	32	0	0	0	0	0	[NONE]	[NONE]	[NONE]	HEX
10	DESTINATION ADDRESS	16	32	0	0	0	0	0	[NONE]	[NONE]	[NONE]	HEX

FIG. 5A

HEADER LENGTH LOOKUP STRUCTURE					
PROTOCOL	NEXT INDEX	MINIMUM	MAXIMUM	MASK	TRANSLATION
[NONE]	11	0x0	0x4	ALL	"INVALID LENGTH"
[NONE]	2	0x5	0xF	ALL	"BYTES"

FIG. 5B

FRAME TYPE NEXT PROTOCOL STRUCTURE					
PROTOCOL	NEXT INDEX	MINIMUM	MAXIMUM	MASK	TRANSLATION
[NONE]	4	0x00	0x00	ALL	"ILLEGAL PROTOCOL"
GP1	4	0x01	0x01	ALL	"GP1"
GP2	5	0x02	0x02	ALL	"GP2"
[NONE]	4	0x03	0xFF	ALL	"ILLEGAL PROTOCOL"

FIG. 5C

SOURCE SOCKET NEXT PROTOCOL STRUCTURE					
PROTOCOL	NEXT INDEX	MINIMUM	MAXIMUM	MASK	TRANSLATION
[NONE]	8	0x0000	0x0142	ALL	"UNKNOWN PROTOCOL"
GP3	8	0x0143	0x018F	ODD	"GP3"
GP4	8	0x0143	0x018F	EVEN	"GP4"
[NONE]	8	0x0190	0xFFFF	ALL	"ILLEGAL PROTOCOL"

FIG. 5D

DESTINATION SOCKET NEXT PROTOCOL STRUCTURE					
PROTOCOL	NEXT INDEX	MINIMUM	MAXIMUM	MASK	TRANSLATION
[NONE]	9	0x0000	0x0142	ALL	"UNKNOWN PROTOCOL"
GP3	9	0x0143	0x018F	ODD	"GP3"
GP4	9	0x0143	0x018F	EVEN	"GP4"
[NONE]	9	0x0190	0xFFFF	ALL	"ILLEGAL PROTOCOL"

FIG. 5E

GP MASTER OPTION CONTROL RECORD					
PROTOCOL NAME	NUMBITS	NUMFIELDS	CURFIELD	FIELDS	OPTIONS
GP MASTER OPTION	0	1	0	FIGURE 6A	[NONE]

FIG. 6

INDEX	FIELD NAME	BIT OFFSET	BIT LENGTH	LEFT SHIFT	RIGHT SHIFT	CHECK SUM	FRAME LENGTH	HEADER LENGTH	STATISTICS	LOOKUP STRUCTURE	FILTER	FORMAT
0	[NONE]	0	0	0	24	0	0	0	[NONE]	FIGURE 6B	[NONE]	[NONE]

FIG. 6A

VENDOR ADDRESS LOOKUP STRUCTURE					
PROTOCOL	NEXT INDEX	MINIMUM	MAXIMUM	MASK	TRANSLATION
FIGURE 7	1	0x00	0x00	ALL	"GP EOL OPTION"
FIGURE 8	1	0x01	0x01	ALL	"GP NOOP OPTION"
FIGURE 9	1	0x02	0x02	ALL	"GP MAXSIZE OPTION"
[NONE]	1	0x03	0xFF	ALL	"UNKNOWN OPTION"

FIG. 6B

GP EOL OPTION CONTROL RECORD					
PROTOCOL NAME	NUMBITS	NUMFIELDS	CURFIELD	FIELDS	OPTIONS
GP MASTER OPTION	8	1	0	FIGURE 7A	[NONE]

FIG. 7

INDEX	FIELD NAME	BIT OFFSET	BIT LENGTH	LEFT SHIFT	RIGHT SHIFT	CHECK SUM	FRAME LENGTH	HEADER LENGTH	STATISTICS	LOOKUP STRUCTURE	FILTER	FORMAT
0	[NONE]	0	8	0	24	0	0	0	[NONE]	FIGURE 7B	[NONE]	HEX

FIG. 7A

EOL LOOKUP STRUCTURE					
PROTOCOL	NEXT INDEX	MINIMUM	MAXIMUM	MASK	TRANSLATION
FIGURE 6	1	0x00	0x00	ALL	"EOL"

FIG. 7B

GP NOOP OPTION CONTROL RECORD					
PROTOCOL NAME	NUMBITS	NUMFIELDS	CURFIELD	FIELDS	OPTIONS
GP NOOP OPTION	8	1	0	FIGURE 8A	[NONE]

FIG. 8

INDEX	FIELD NAME	BIT OFFSET	BIT LENGTH	LEFT SHIFT	RIGHT SHIFT	CHECK SUM	FRAME LENGTH	HEADER LENGTH	STATISTICS	LOOKUP STRUCTURE	FILTER	FORMAT
0	[NONE]	0	8	0	24	0	0	0	[NONE]	FIGURE 8B	[NONE]	HEX

FIG. 8A

NOOP LOOKUP STRUCTURE					
PROTOCOL	NEXT INDEX	MINIMUM	MAXIMUM	MASK	TRANSLATION
FIGURE 6	1	0x01	0x01	ALL	"NOOP"

FIG. 8B

GP MINMAXSIZE OPTION CONTROL RECORD					
PROTOCOL NAME	NUMBITS	NUMFIELDS	CURFIELD	FIELDS	OPTIONS
MAXSIZE OPTION	48	4	0	FIGURE 9A	[NONE]

FIG. 9

INDEX	FIELD NAME	BIT OFFSET	BIT LENGTH	LEFT SHIFT	RIGHT SHIFT	CHECK SUM	FRAME LENGTH	HEADER LENGTH	STATISTICS	LOOKUP STRUCTURE	FILTER	FORMAT
0	OPTION TYPE	0	8	0	24	0	0	0	[NONE]	FIGURE 9B	[NONE]	HEX
1	OPTION LENGTH	0	8	8	24	0	0	8	[NONE]	[NONE]	[NONE]	HEX
2	MINSIZE	0	16	16	16	0	0	0	[NONE]	[NONE]	[NONE]	DECIMAL
3	MAXSIZE	4	16	0	16	0	0	0	[NONE]	[NONE]	[NONE]	DECIMAL

FIG. 9A

MINMAXSIZE LOOKUP STRUCTURE					
PROTOCOL	NEXT INDEX	MINIMUM	MAXIMUM	MASK	TRANSLATION
FIGURE 6	1	0x02	0x02	ALL	"MINMAXSIZE"

FIG. 9B

FILTER CHANNEL CONTROL STRUCTURE		
SYSTEM FILTER STATUS	NUMBER OF FILTERS	POINTER TO FILTER CHANNELS
FILTER_FRAME	1	FIGURE 10A

FIG. 10

FILTER CHANNELS				
INDEX	NEXTCRITERIAINDEX	TOTALCRITERIA	CRITERIA POINTER	CHANNEL NAME
0	0	3	FIGURE 10B	GP TYPE OR STATION

FIG. 10A

FILTER CHANNELS				
INDEX	CHANNEL PTR	LOOKUP POINTER	PROTOCOL POINTER	FIELD POINTER
0	FIGURE 10A, INDEX 0	FIGURE 10C	FIGURE 4	FIGURE 4A, INDEX 0
1	FIGURE 10A, INDEX 0	FIGURE 10D	FIGURE 4	FIGURE 4A, INDEX 1
2	FIGURE 10A, INDEX 0	FIGURE 10E	FIGURE 5	FIGURE 5A, INDEX 3

FIG. 10B

INDEX 0 FILTER CONDITION LOOKUP STRUCTURE					
RETURN VALUE	NETINDEX	MINIMUM	MAXIMUM	MASK	TRANSLATION
FILTER_FRAME	2	0x000000	0x08FFFE	ALL	""
FILTER_FRAME	1	0x08FFFF	0x08FFFF	ALL	"VENDOR XXX"
FILTER_FRAME	2	0x090000	0xFFFFFF	ALL	""

FIG. 10C

INDEX 1 FILTER CONDITION LOOKUP STRUCTURE					
RETURN VALUE	NETINDEX	MINIMUM	MAXIMUM	MASK	TRANSLATION
FILTER_FRAME	2	0x000000	0x334454	ALL	""
PASS_FRAME	3	0x334455	0x334455	ALL	"334455"
FILTER_FRAME	2	0x334456	0xFFFFFF	ALL	""

FIG. 10D

INDEX 2 FILTER CONDITION LOOKUP STRUCTURE					
RETURN VALUE	NETINDEX	MINIMUM	MAXIMUM	MASK	TRANSLATION
FILTER_FRAME	3	0x00	0x00	ALL	""
PASS_FRAME	3	0x01	0x02	ALL	"GP1 OR GP2"
FILTER_FRAME	3	0x03	0xFF	ALL	""

FIG. 10E

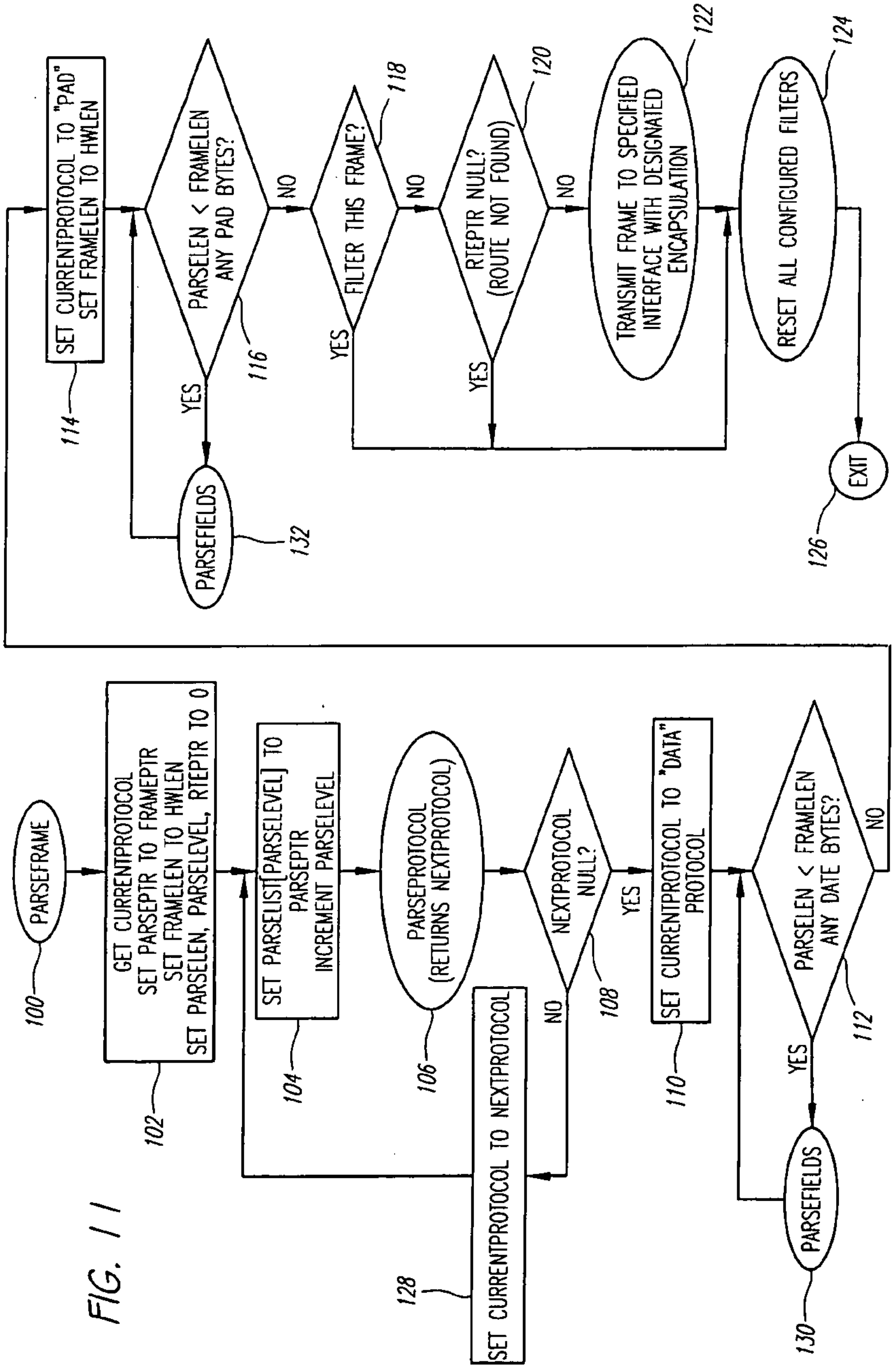


FIG. 11
100
PARSEFRAME
102
GET CURRENTPROTOCOL
SET PARSEPTR TO FRAMEPTR
SET FRAMELEN TO HWLEN
SET PARSELEN, PARSELEVEL, RTEPTR TO 0
104
SET PARSELIST[PARSELEVEL] TO PARSEPTR
INCREMENT PARSELEVEL
106
PARSEPROTOCOL (RETURNS NEXTPROTOCOL)
108
NEXTPROTOCOL NULL?
NO
YES
128
SET CURRENTPROTOCOL TO NEXTPROTOCOL
110
SET CURRENTPROTOCOL TO "DATA" PROTOCOL
112
PARSELEN < FRAMELEN ANY DATE BYTES?
YES
NO
130
PARSEFIELDS
114
SET CURRENTPROTOCOL TO "PAD"
SET FRAMELEN TO HWLEN
116
PARSELEN < FRAMELEN ANY PAD BYTES?
YES
NO
132
PARSEFIELDS
118
FILTER THIS FRAME?
YES
NO
120
RTEPTR NULL? (ROUTE NOT FOUND)
YES
NO
122
TRANSMIT FRAME TO SPECIFIED INTERFACE WITH DESIGNATED ENCAPSULATION
124
RESET ALL CONFIGURED FILTERS
126
EXIT

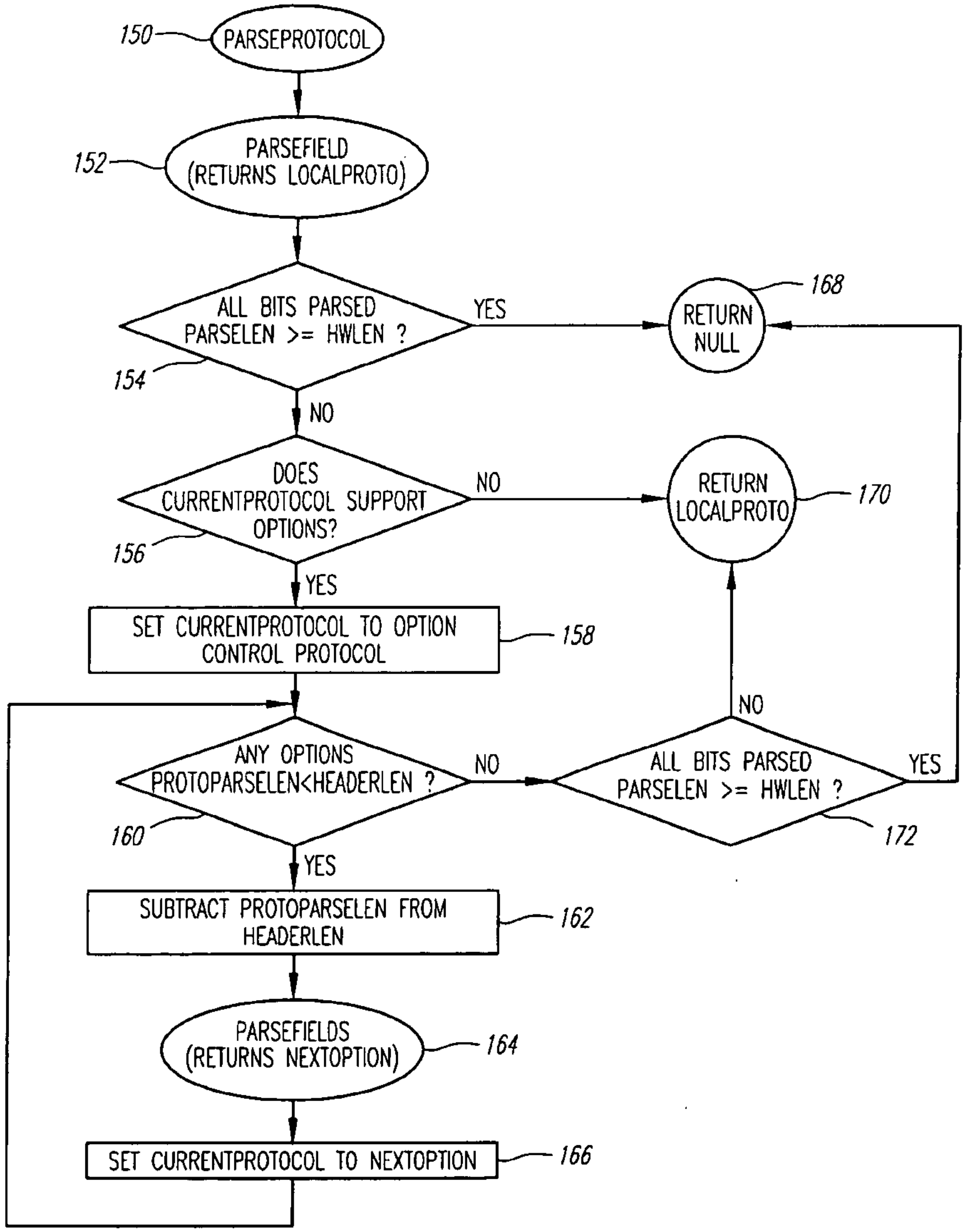

150
PARSEPROTOCOL
152
PARSEFIELD (RETURNS LOCALPROTO)
ALL BITS PARSED PARSELEN >= HWLEN ?
154
YES
RETURN NULL
168
NO
DOES CURRENTPROTOCOL SUPPORT OPTIONS?
156
NO
RETURN LOCALPROTO
170
YES
SET CURRENTPROTOCOL TO OPTION CONTROL PROTOCOL
158
ANY OPTIONS PROTOPARSELEN<HEADERLEN ?
160
NO
ALL BITS PARSED PARSELEN >= HWLEN ?
172
NO
YES
YES
SUBTRACT PROTOPARSELEN FROM HEADERLEN
162
PARSEFIELDS (RETURNS NEXTOPTION)
164
SET CURRENTPROTOCOL TO NEXTOPTION
166

FIG. 12

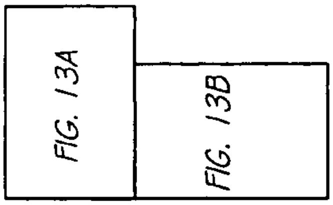


FIG. 13

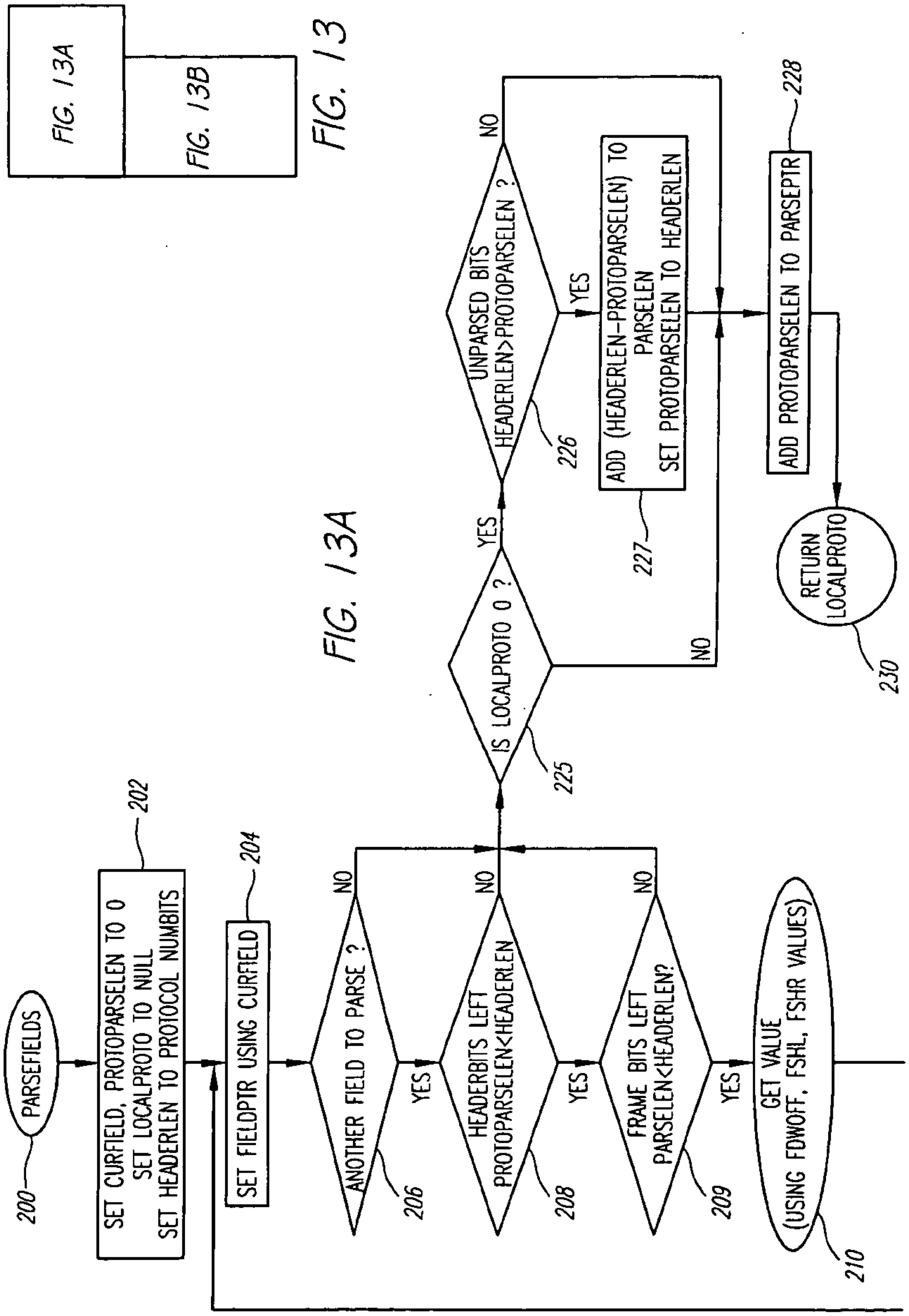


FIG. 13A

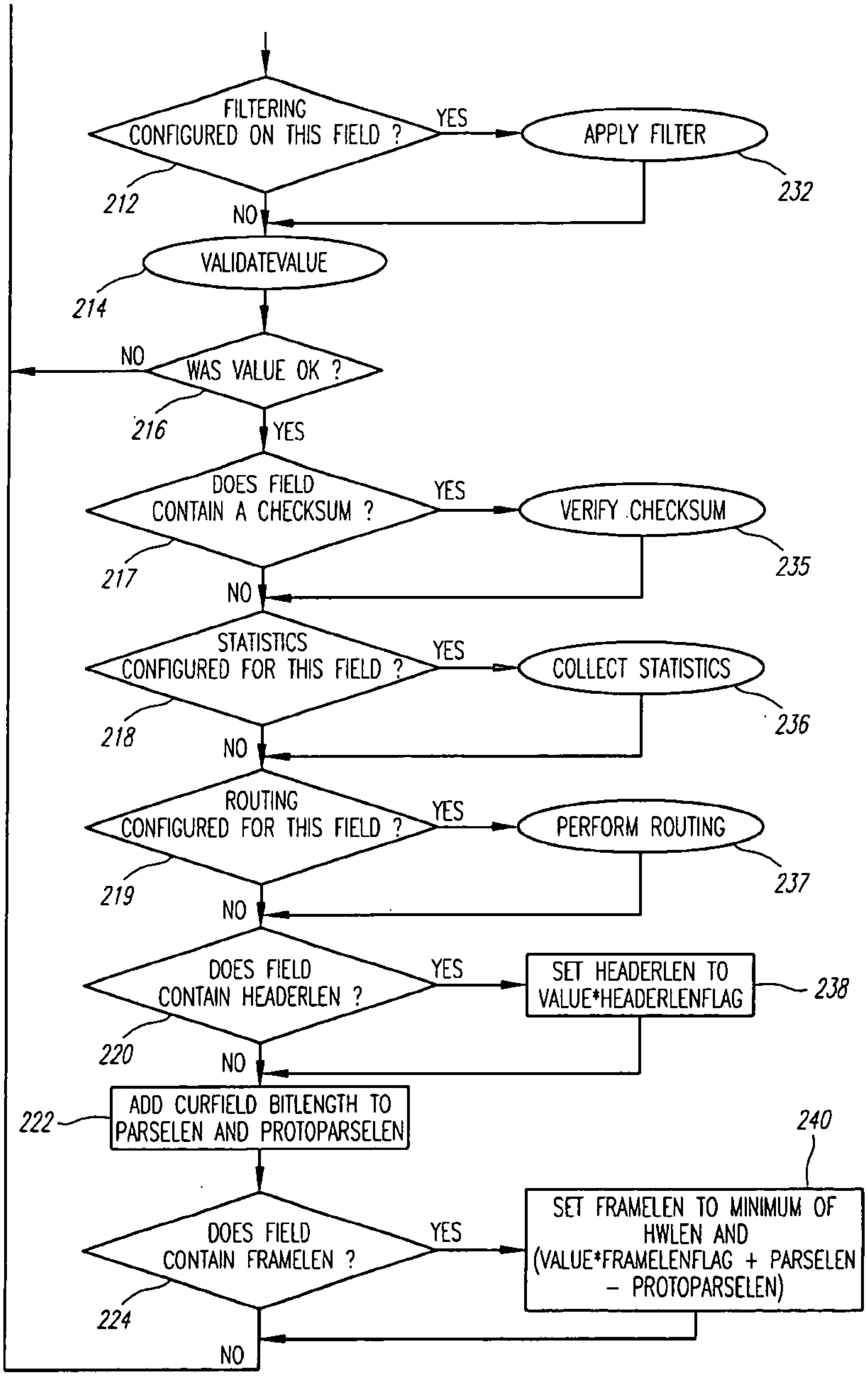

FILTERING CONFIGURED ON THIS FIELD ?
YES
APPLY FILTER
232
212
NO
VALIDATEVALUE
214
NO
WAS VALUE OK ?
216
YES
DOES FIELD CONTAIN A CHECKSUM ?
YES
VERIFY CHECKSUM
235
217
NO
STATISTICS CONFIGURED FOR THIS FIELD ?
YES
COLLECT STATISTICS
236
218
NO
ROUTING CONFIGURED FOR THIS FIELD ?
YES
PERFORM ROUTING
237
219
NO
DOES FIELD CONTAIN HEADERLEN ?
YES
SET HEADERLEN TO VALUE*HEADERLENFLAG
238
220
NO
222
ADD CURFIELD BITLENGTH TO PARSELEN AND PROTOPARSELEN
240
DOES FIELD CONTAIN FRAMELEN ?
YES
SET FRAMELEN TO MINIMUM OF HWLEN AND (VALUE*FRAMELENFLAG + PARSELEN − PROTOPARSELEN)
224
NO

FIG. 13B

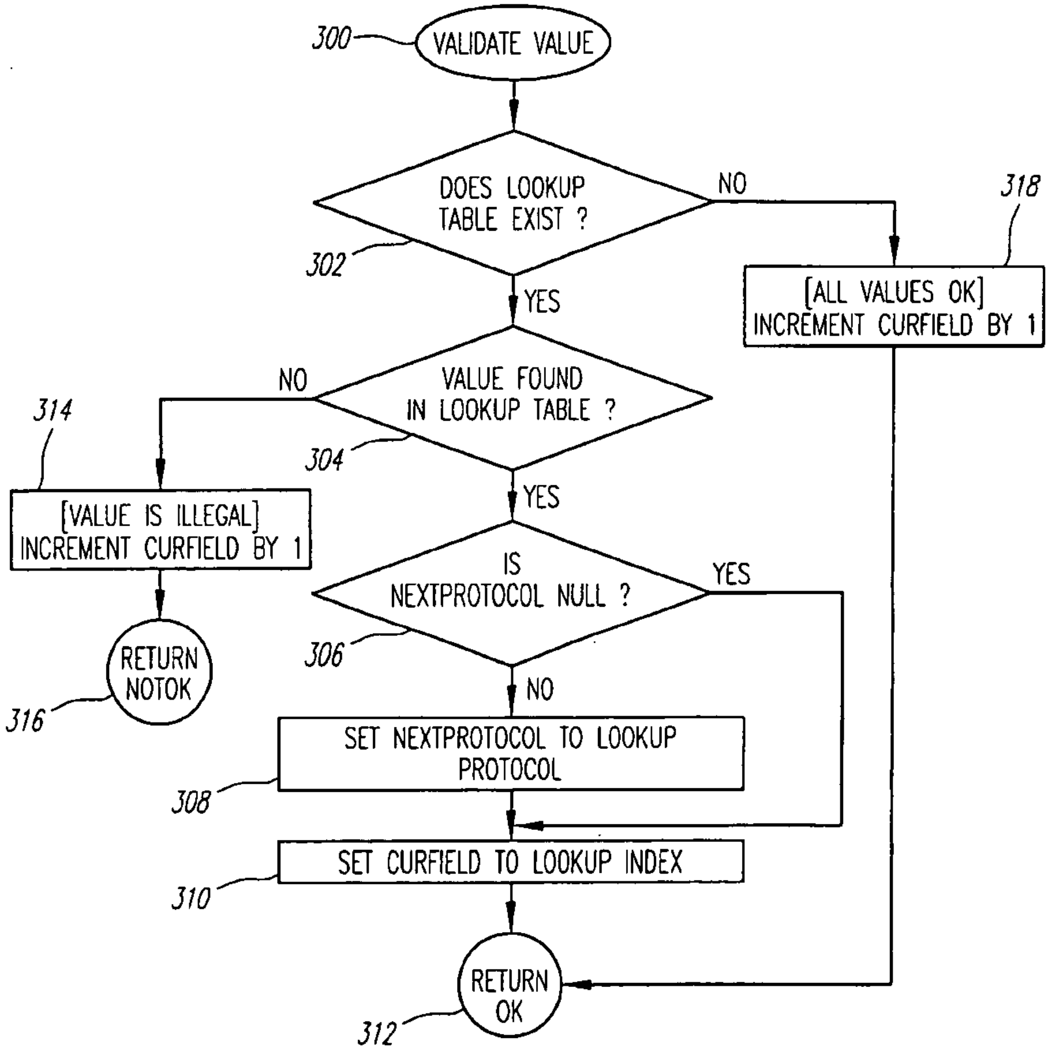

300
VALIDATE VALUE
DOES LOOKUP TABLE EXIST ?
302
NO
318
[ALL VALUES OK] INCREMENT CURFIELD BY 1
YES
VALUE FOUND IN LOOKUP TABLE ?
NO
304
314
[VALUE IS ILLEGAL] INCREMENT CURFIELD BY 1
YES
IS NEXTPROTOCOL NULL ?
YES
306
RETURN NOTOK
316
NO
SET NEXTPROTOCOL TO LOOKUP PROTOCOL
308
SET CURFIELD TO LOOKUP INDEX
310
RETURN OK
312

FIG. 14

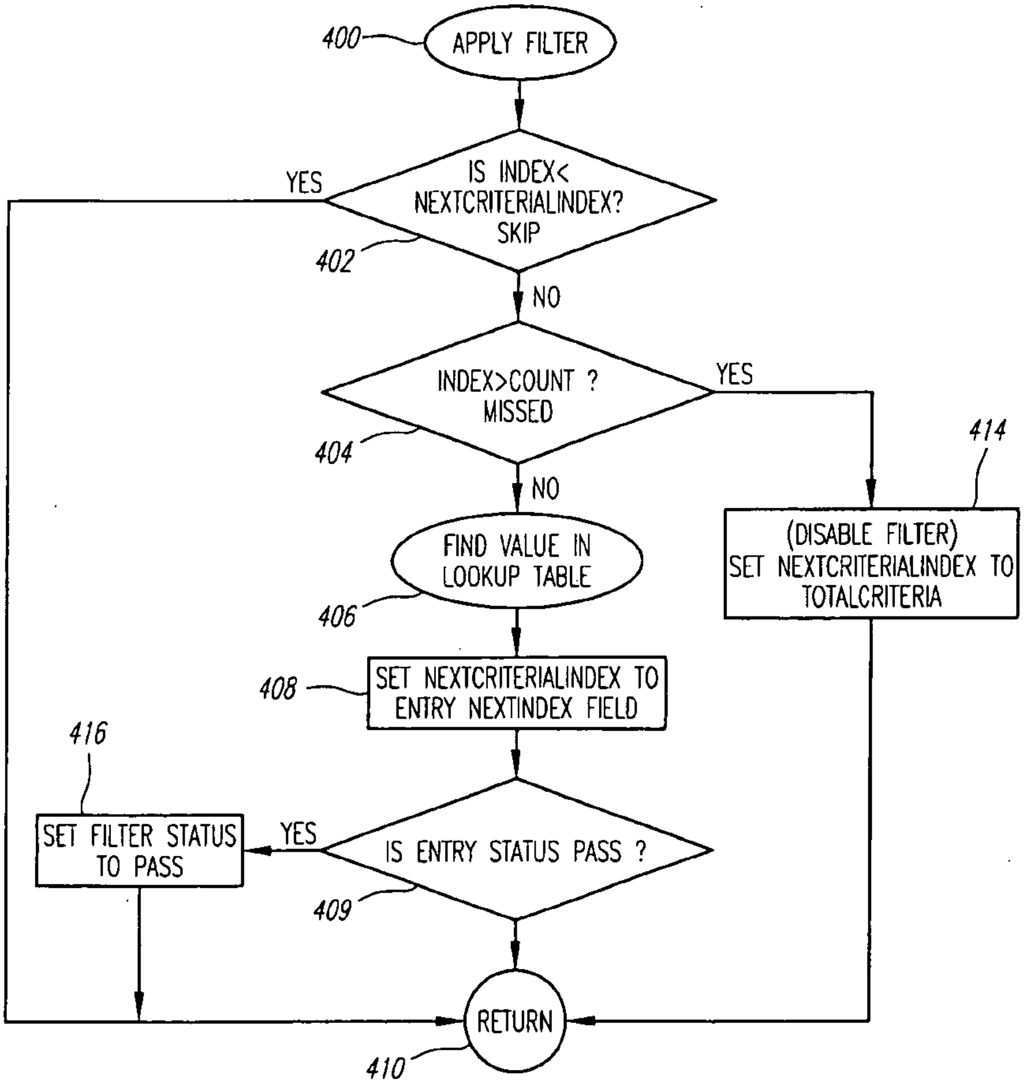

400
APPLY FILTER
IS INDEX<
NEXTCRITERIALINDEX?
SKIP
402
YES
NO
INDEX>COUNT ?
MISSED
404
YES
NO
414
FIND VALUE IN
LOOKUP TABLE
406
(DISABLE FILTER)
SET NEXTCRITERIALINDEX TO
TOTALCRITERIA
408
SET NEXTCRITERIALINDEX TO
ENTRY NEXTINDEX FIELD
416
SET FILTER STATUS
TO PASS
YES
IS ENTRY STATUS PASS ?
409
RETURN
410

FIG. 15

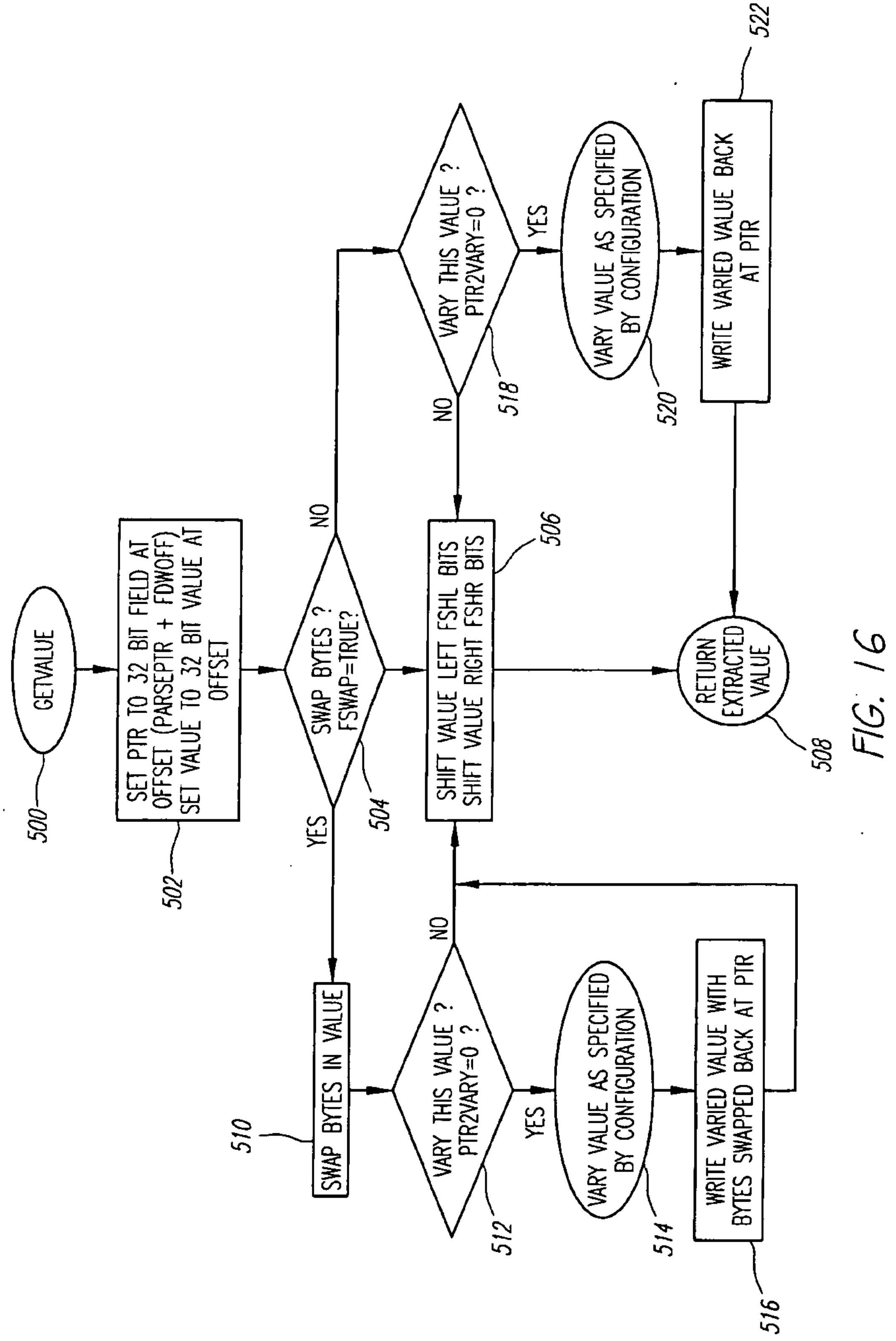

GETVALUE
500
SET PTR TO 32 BIT FIELD AT OFFSET (PARSEPTR + FDWOFF) SET VALUE TO 32 BIT VALUE AT OFFSET
502
SWAP BYTES ? FSWAP=TRUE?
504
YES
NO
SHIFT VALUE LEFT FSHL BITS SHIFT VALUE RIGHT FSHR BITS
506
RETURN EXTRACTED VALUE
508
VARY THIS VALUE ? PTR2VARY=0 ?
518
NO
YES
VARY VALUE AS SPECIFIED BY CONFIGURATION
520
WRITE VARIED VALUE BACK AT PTR
522
SWAP BYTES IN VALUE
510
VARY THIS VALUE ? PTR2VARY=0 ?
512
NO
YES
VARY VALUE AS SPECIFIED BY CONFIGURATION
514
WRITE VARIED VALUE WITH BYTES SWAPPED BACK AT PTR
516

FIG. 16

NETWORK FILTERING SYSTEM

This is a continuation of application Ser. No. 08/888,875, filed Jul. 7, 1997, now U.S. Pat. No. 5,781,729; which is a continuation of Ser. No. 08/575,506, filed Dec. 20, 1995, now U.S. Pat. No. 5,793,954.

TECHNICAL FIELD

The present invention relates to network communications systems and, in particular, to improved systems and methods for parsing, filtering, generating and analyzing data composed of inter-related structures such as protocols found within network frames.

The application includes a microfiche appendix of software developed applicable to the system disclosed, consiting of one (1) slide and 84 frames.

BACKGROUND ART

Existing network interface devices provide systems for receiving, analyzing, filtering and transmitting network data or frames of data. Network Protocol Analyzers, Bridges, and Routers are among the most common network interface devices currently available.

Conventional network protocol analyzers provide, for a predefined set of network frame structures or protocols, a system for monitoring the activity of a network and the stations on it by allowing network traffic to be captured and stored for later analysis. Common capture and analysis capabilities include the gathering of statistics, subsequent report generation, the ability to filter frames based on specific criteria, and the ability to generate network traffic.

Bridges and routers are network devices that pass frames from one network interface to another. Bridges operate at the data-link layer and routers at the network layer of the OSI reference model. Like protocol analyzers, both bridges and routers may gather statistics and filter incoming network frames based on specific criteria, however incoming frames also may be forwarded to other networks based on information collected by the bridge or router. Routers typically support only a limited number of network protocols.

Each of these network devices requires an ability to separate network frames into individual protocols and their components (typically referred to as parsing), an ability to filter incoming frames based on a logical combination of one or more field values extracted during parsing, and an ability to gather statistics based in part on extracted field values. Typically, it is a requirement that network frames be received, analyzed and forwarded at full network speeds, sometimes on many different networks at one time.

A frame filter consists of one or more criteria which specify one or more valid values for a frame (or segments of a frame). Frame filtering criteria are typically implemented using an offset (from frame or protocol header start), a length in bits which defines a field, a value for comparison, and mask values for identifying relevant and irrelevant bits within the field. For multiple value filter criteria, the result from each filter value is logically OR'ed together to obtain an overall result. Therefore, each additional result adds to the processing required to filter a given field. For filtering on optional protocol fields that do not occur at the same relative offset in each protocol frame, this method is time-consuming. Thus, it would be desirable to perform filtering on both fixed and optional variable offset fields for any number of values or ranges of values without incurring any additional overhead.

Parsing, the process wherein network frames are broken up into their individual protocols and fields, is necessary for filtering with offsets relative to protocol headers, gathering field based statistics, generating network traffic, routing data frames, verifying field values, and displaying network frames in human readable form. In conventional systems, the parsing process has an overall structure which incorporates control logic for each supported protocol. Therefore, additional control logic must be developed when support for a new protocol is added to a conventional system. As the development of additional control logic, whether implemented in hardware or software, may be both time consuming and expensive, it would be highly desirable to be able to parse all protocols with a single configurable software (or hardware) module so that support for additional protocols could be added to a system without requiring substantial modification to the system or its control logic.

Further, although microprocessors (or CPUs) available today can execute tens or even hundreds of millions of instructions per second, vendors often must provide dedicated hardware assistance and/or front-end processors with hand-coded assembly language routines to achieve the necessary processing rates for more than one pair of networks. Unfortunately, this solution requires hardware and/or software modifications whenever changes are made to the number of supported features or protocols.

Finally, as networks become larger and more complex, the maintenance of a comprehensive statistics database by each network device becomes more important. Because these statistics databases typically are not utilized by a maintaining device, but instead are collected by a network management device, the collection process may affect performance adversely without any corresponding benefit to the collecting device.

In light of the considerations discussed above, it is believed that a network interface system having a configurable protocol analysis capability with common control logic applicable to many different network devices would be highly desirable.

SUMMARY OF INVENTION

The present invention is directed to improved systems and methods for parsing, filtering, generating and analyzing data (or frames of data) transmitted over a data communications network. In one particularly innovative aspect of the present invention, a single logic control module, which may be implemented in hardware or software, is utilized to perform any of a number of data manipulation functions (for example, parsing, filtering, data generation or analysis functions) based upon one or more programmably configurable protocol descriptions which may be stored in and retrieved from an associated memory.

The use of common control logic (i.e. the use of a single logic control module) and programmably configurable protocol descriptions allows changes to existing protocols to be made and support for new protocols to be added to a system in accordance with the present invention through configuration only—without the need for hardware and/or software system modifications. Thus, those skilled in the art will appreciate that a network interface in accordance with the present invention may be configured and reconfigured, if necessary, in a highly efficient and cost effective manner to implement numerous data manipulation functions and to accommodate substantial network modifications (for example, the use of different data transmission hardware, protocols or protocol suites) without necessitating substantial system changes.

In one preferred form, the system of the present invention may employ a CPU or other hardware implementable method for analyzing data from a network in response to selectively programmed parsing, filtering, statistics gathering, and display requests. Moreover, the system of the present invention may be incorporated in a network device, such as a network analyzer, bridge, router, or traffic generator, including a CPU and a plurality of input devices, storage devices, and output devices, wherein frames of network data may be received from an associated network, stored in the storage devices, and processed by the CPU based upon one or more programmably configurable protocol descriptions also stored in the storage devices. The protocol descriptions may take the form of one or more protocol description files for each supported network protocol and may include a protocol header record and plurality of field sub-records having data corresponding to an associated protocol and fields defined therein.

The system of the present invention also preferably includes logic for extracting field values from particular network frames, performing validation and error checking, and making parsing decisions based upon field values and information in the programmably configurable protocol descriptions.

The system of the present invention also preferably includes logic for filtering a subset of network frames received from the input or storage devices which satisfy a filter criteria based upon information defined in the programmably configurable protocol descriptions.

The system of the present invention also preferably includes logic for filtering network frames which satisfy a plurality of filter criteria which, if desired, may be joined together by Boolean operators.

The system of the present invention also preferably includes logic for analyzing a filter request by breaking the request into its component criteria to determine whether the result from evaluating a particular filter request criteria when combined with results from earlier criteria can be used to filter (i.e. discard) a particular network frame.

The system of the present invention also preferably includes logic for collecting statistics based upon extracted field values satisfying a statistics criteria based upon information defined in the programmably configurable protocol descriptions.

The system of the present invention also preferably includes logic for determining a next protocol description structure required to continue analyzing a network frame.

The system of the present invention also preferably includes logic for determining a frame length and individual protocol header lengths from extracted field values in a network frame.

The system of the present invention also preferably includes logic for making routing decisions based upon information contained in the programmably configurable protocol descriptions.

The system of the present invention also preferably includes logic for determining display formats based on information contained in the programmably configurable protocol descriptions.

The system of the present invention also preferably includes logic for verifying individual field values and making parsing decisions based on the validity of the value.

The system of the present invention also preferably includes logic for constructing and transmitting network frames with varying field contents based on information contained in the programmably configurable protocol descriptions.

The system of the present invention may be employed in any system where it is useful to be able to examine and perform various operations on contiguous bit-fields in data structures, wherein each data structure is composed of predefined fields of one or more contiguous bits. Further, the system of the present invention is particularly efficient where operations must be performed on a subset of included fields.

Those skilled in the art will recognize that the system of the present invention gains a distinct advantage in size and maintainability over conventional network devices by implementing analysis capabilities for multiple known and unknown protocols using common control logic. Furthermore, the system gains a distinct advantage in speed and efficiency over conventional network devices when the control logic is implemented in hardware or a front-end processor, without incurring the penalty of additional hardware and/or software development when protocol definitions change.

Accordingly, it is an object of the present invention to provide an improved system for network analysis wherein the system may determine which protocols and which protocol fields exist in a network frame (also referred herein as parsing) using common control logic combined with configurable protocol descriptions.

It is yet another object of the present invention to provide an improved system for network analysis wherein the control logic may be implemented in hardware as well as software.

It is yet another object of the present invention to provide an improved system for network analysis wherein each supported analysis capability is configurable even when the control logic is implemented in hardware.

It is another object of the present invention to provide an improved system for network analysis wherein the system may determine whether a particular network frame includes a field that satisfies a particular filter criteria based upon information stored in a programmably configurable protocol description.

It is yet another object of the present invention to provide an improved system for network analysis wherein the system may determine if a particular network frame includes a protocol field that satisfies a particular statistics gathering criteria defined in a programmably configurable protocol description.

It is yet another object of the present invention to provide an improved system for network analysis wherein the system may generate network traffic in the form of frames constructed from selected protocol descriptions with the ability to specify a variety of methods for varying individual field values.

It is still another object of the present invention to provide an improved system for network analysis wherein the system may route network frames (determine the appropriate destination interface) that satisfy a particular routing criteria defined in a programmably configurable protocol description while providing a capability to specify a variety of methods for varying individual field values during the routing process.

It is still another object of the present invention to provide an improved system for network analysis wherein the system may determine if a particular network frame includes a protocol field that contains a value related to either the overall length of the frame or the current protocol header length.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram of a network interface system in accordance with one form of the present invention.

FIG. 2 is a diagram representing a set of data records of a typical network frame which may be contained in the data files of the network interface system illustrated in FIG. 1.

FIG. 3 is a diagram representing a set of data records of a protocol description in accordance with one form of the present invention.

FIG. 4 is a diagram representing a control record of an Ethernet protocol description which may be utilized in a network interface system in accordance with one form of the present invention.

FIG. 4*a* is a diagram representing five defined field sub-records of the Ethernet protocol description illustrated in FIG. 4.

FIGS. 4*b*, 4*c*, and 4d are diagrams representing lookup structures referenced in FIG. 4*a* fields 0, 2 and 4 respectively.

FIG. 5 is a diagram representing a control record of an imaginary Generic Protocol description which may be utilized in a network interface system in accordance with one form of the present invention.

FIG. 5*a* is a diagram representing eleven defined field sub-records of the GP description illustrated in FIG. 5.

FIGS. 5*b*, 5*c*, 5*d*, and 5*e* are diagrams representing lookup structures referenced in FIG. 5(*a*) fields 1, 3, 7 and 8, respectively.

FIGS. 6, 6*a*, and 6*b* are diagrams representing the control record and field sub-record of a protocol description structure that allows parsing of optional fields of the GP description shown in FIGS. 5–5*e*.

FIGS. 7, 7*a*, and 7*b* are diagrams representing the control record and field sub-records of a protocol description structure that describes the End Of List option of the GP description shown in FIGS. 5–5*e*.

FIGS. 8, 8*a*, and 8*b* are diagrams representing the control record and field sub-records of a protocol description structure that describes the No Operation option of the GP description shown in FIGS. 5–5*e*.

FIGS. 9, 9*a*, and 9*b* are diagrams representing the control record and field records of a protocol description file that describes the Maximum Frame Size option of the GP description shown in FIGS. 5–5*e*.

FIGS. 10, 10*a*, 10*b*, 10*c*, 10*d* and 10*e* are diagrams representing data records of a filter expression control and associated field filter structures.

FIG. 11 is a flow chart illustrating top level frame parsing control logic in accordance with one form of the present invention.

FIG. 12 is a flow chart illustrating protocol parsing control logic in accordance with one form of the present invention.

FIG. 13 is a flow chart of the field parsing control logic in accordance with one form of the present invention.

FIG. 14 is a flow chart representing value verification, error checking, next protocol and branch determination control logic in accordance with one form of the present invention.

FIG. 15 is a flow chart representing field filtering control logic in accordance with one form of the present invention.

FIG. 16 is a flow chart illustrating field value extraction and varying control logic in accordance with one form of the present invention.

DESCRIPTION OF PREFERRED EMBODIMENTS

Referring now to FIG. 1, a network interface system in accordance with one form of the present invention, generally referred to as 10, may be implemented in a network device including input devices 12, data storage devices 14, analysis control logic 16 for facilitating the input, storage, retrieval, and analysis of network frames, and output devices 18 for forwarding frames or displaying or printing the results of analyses. A data storage device 14 may include a data file 20 of network frames having n protocol data records, wherein each data record contains data stored in a plurality of predefined fields. Protocol description files 22 also may be stored in the data storage device 14. The protocol description files 22 may include a protocol control record and n field sub-records, which together may describe a subset of a network protocol and include rules for analyzing that protocol.

The network device control logic 16 is capable of retrieving a subset of network frames from the input devices 12 or data files 20 which satisfy one or more criteria based upon extracted field values and filtering criteria contained in one or more of the protocol description files 22. The network device control logic 16 also includes logic for determining frame and protocol header lengths, gathering statistics, verification and error checking, determining routes, varying values, and formatting output.

A personal computer or conventional network device, such as an IBM PC (or compatible), Apple Macintosh®, or any Unix®, or Zenix® workstation, protocol analyzer, bridge, router, traffic generator, or similar system may be utilized in accordance with the system of the present invention. The data input devices 12 may comprise any of a number of commercially available network interface devices and may include a conventional keyboard or mouse if required. The data storage devices 14 may take the form of any of a number of commercially available data storage options (such as RAM, ROM, EPROM, or various sized fixed disk drives), and the data output devices 18 may comprise any of a number of commercially available user interface devices, such as CRT displays, monitors, network interface devices and/or printers (if required). The analysis control logic 16 may be implemented as a computer program written in any language suitable for systems programming or may be implemented in hardware if better performance is required. In one presently preferred form, the analysis control logic 16 may be implemented via the programming files set forth in the attached Appendix, which is herein incorporated by reference. However, those skilled in the art will appreciate that the analysis control logic 16 might equivalently be implemented in dedicated hardware using, for example, one or more application specific integrated circuits ("ASICs") or one or more field programmable gate arrays ("FPGAs").

The network interface system 10 of the present invention is preferably implemented on a personal computer, workstation or conventional network device having a 32-bit or larger bus and register set, an optional math co-processor, at least one megabyte of available RAM, and for personal computer and workstation applications, a fixed disk having at least 10 megabytes of available storage space. As shown in the attached Appendix, the analysis control logic 16 may be programmed in the C++ language, with abstract data types defined for statistics gathering, value verification, next protocol determination, filtering, varying values, checksumming and route determination capabilities, and protocol control and field records.

Referring now to FIG. 2, a data file 20 in accordance with one form of the present invention may include a plurality (n) of protocol header data records and optional Data and Pad records. Each protocol record contains data organized into a plurality of predefined fields. Each field comprises a collection of 1 or more contiguous bits and includes a set of valid values for that field. For example, a particular protocol specification might include a 6 bit header length field that limits the protocol header length to values between 20 and 60 inclusive, thereby excluding values less than 20 and values from 61 to 64.

The number of possible contiguous bit fields for a protocol header of length N bits where N is greater than 1 can be expressed by the following formula:

$$\text{Number of Possible Fields} = \sum_{i=1}^{N} i$$

It will be appreciated by those skilled in the art that any possible organization of fields for any possible protocol specification is contemplated for the network interface system **10** of the present invention.

Referring now to FIG. **3**, a protocol description file **22** in accordance with one form of the present invention may include a protocol control record, and a plurality (n) of field data records. In a particularly preferred embodiment, the protocol control record (shown below in Table 1) may define the overall structure of a network protocol and reference other information relating to the network protocol.

TABLE 1

PROTOCOL CONTROL RECORD

Offset	Name	Description
0–3	name_length	length of protocol name in bytes including NULL terminator
4–7	protocol_name	name of protocol control record is describing
8–11	filename	name of file control record is stored in
12–15	numBits	total bit length of protocol header control record is describing
16–17	numFields	number of fields required to describe protocol header
18–19	curField	index of field currently referenced
20–23	outFlag	flag indicating template has been output to file
24–27	dbW	display bit width for protocol header display
28–31	fields	field records that describe protocol header
32–25	options	pointer to option control record to use if this protocol has optional fields
36–39	Rt	pointer to protocol specific routing table

The field records referenced at bytes 28–31 in the table above are preferably organized as shown in Table 2:

TABLE 2

FIELD SUB-RECORDS

Offset	Name	Description
0–3	fplen	flag indicating value is actual length of frame (multiplier)
4–7	fblen	length of field in bits
8–11	fdwoff	byte offset from start of protocol header of 32-bit field containing value
12	fshl	number of bits to left shift 32-bit value
13	fshr	number of bits to right shift 32-bit value
14	ffmt	number indicating a display type (i.e., decimal, hex, . . .)
15	flflag	flag indicating value is actual length of protocol header (multiplier)
16	reserved	not used . . . pad byte to align following fields
17		
	fmult	multiplier to apply to value prior to display

TABLE 2-continued

FIELD SUB-RECORDS

Offset	Name	Description
18	fswap	flag indicating the need to swap bytes and words in 32-bit field containing value
19	fsdspfield	flag indicating that this field should be displayed
20–23	fname	pointer to field name (0 = none)
24–27	ptr2stats	pointer to configured statistics structure/class (0 = none)
28–31	ptr2np	pointer to lookup structure/class . . . next protocol definition to use (0 = none)
32–35	ptr2vary	pointer to vary field value structure/class (0 = none)
36–39	ptr2csum	pointer to checksum structure/class (0 = none)
40–43	ptr2flt	pointer to filter criteria structure (0 = none)
44–47	ptr2rte	pointer to Route Table structure/class (0 = none)

The statistics records referenced in Table 2, above, at bytes 24–27 are preferably organized as shown in Table 3:

TABLE 3

STATISTICS STRUCTURE/CLASS RECORD

Offset	Name	Description
0–3	StatName	pointer to user assigned name for statistic
4–7	Stat	pointer to derived structure/class for accumulating configured statistic

The next protocol lookup records referenced in the field sub-record table (Table 2) at bytes 28–31 are preferably organized as shown in Table 4:

TABLE 4

LOOKUP STRUCTURE RECORD

Offset	Name	Description
0–3	Protocol	pointer to protocol description structure
4–7	Next Index	index of field in protocol description to parse next
8–11	Minimum	minimum acceptable value for this range
12–15	Maximum	maximum acceptable value for this range
16–19	okbits	selects even only, odd only, or all values in range
20–23	Translation	pointer to associated human language equivalent

Lookup structures can be used for determining the next protocol control record to use, terminating protocol processing on illegal values, branching decisions for variable length headers or overlapping fields, and for translation of numeric values to mnemonic or written language equivalents. This ability to specify branches on field values allows protocols with multiple overlapping structures to be specified and parsed dynamically.

The vary field value records referenced in the field sub-record table (Table 2) at bytes 32–35 are preferably organized as shown in Table 5:

TABLE 5

VARY FIELD VALUE RECORD

Offset	Name	Description
0–3	mask	mask for isolating field bits from 32-bit field
4–7	notmask	mask for isolating bits not in field
8–11	operand	value to apply to field bits (relative to field)
12–15	minvalue	minimum allowable value for field bits (relative to field)

TABLE 5-continued

VARY FIELD VALUE RECORD

Offset	Name	Description
16–19	maxvalue	maximum allowable value for field bits (relative to field)

The checksum records referenced in the field sub-record table (Table 2) at bytes 36–39 are preferably organized as shown in Table 6:

TABLE 6

CHECKSUM RECORD

Offset	Name	Description
0–3	verify	pointer to routine to verify protocol checksum
4–7	compute	pointer to routine to compute protocol checksum

The filter criteria records referenced in the field sub-record table (Table 2) at bytes 40–43 are preferably organized as shown in Table 7:

TABLE 7

FILTER CRITERIA RECORD

Offset	Name	Description
0–3	Index	index of this filter criteria (zero-based)
4–7	ChPtr	pointer to parent filter channel
8–11	Ranges	pointer to lookup structure containing all possible field values
12–15	Ptl	pointer to associated protocol definition for this criteria
16–19	Fld	pointer to associated field definition for this criteria

The filter channel records referenced in the Filter Criteria record (Table 7) above at 4–7 are preferably organized as shown in Table 8:

TABLE 8

FILTER CHANNEL RECORD

Offset	Name	Description
0–3	NextCriteriaIndex	index of next criteria that should be applied to this filter
4–7	TotalCriteria	number of criteria required to implement this filter
8–11	Criteria	pointer to array of Totalcriteria criteria structures
12–15	ChannelName	pointer to user supplied filter channel name

Each configured filter consists of one or more filter criteria and the filter criteria may be organized into Filter Criteria records. The Filter Criteria records may refer to lookup structures which allow the filter criteria to determine from a field value the current state of the filter expression at each criteria. These states may include: PASS_FRAME (accept this frame) and FILTER_FRAME (discard this frame).

The NextCriteriaIndex field referenced in Table 8 above at bytes 0–3 is used to ensure that all filter expressions are applied in the required order. The Ptl and Fld fields at bytes 12–19 allow filter criteria to be associated with specific protocols and protocol fields. The lookup records referenced in the Filter Criteria record (Table 7) at bytes 8–11 are preferably organized as shown in Table 9:

TABLE 9

FILTER LOOKUP STRUCTURE RECORD

Offset	Name	Description
0–3	Return Value	PASS FRAME, FILTER FRAME value range result
4–7	Index	index of field in Filter Expression structure
8–11	Minimum	minimum acceptable value for this range
12–15	Maximum	maximum acceptable value for this range
16–19	Mask	selects EVEN, ODD or all values in range
20–23	Translation	pointer to associated human language equivalent

The Route Table records referenced in the Field Sub-Records table (Table 2) at bytes 44–47 are preferably organized as shown in Table 10:

TABLE 10

ROUTE TABLE RECORD

Offset	Name	Description
0–11	NetMask	mask for extracting 1 to 96 bits from protocol header route field
12–15	entries	number of entries in Route Table
16–19	Table	pointer to array of 'entries' Route Table entries

The Route Table Entry records referenced in the table above at bytes 16–19 are preferably organized as shown in Table 11:

TABLE 11

ROUTE TABLE ENTRY RECORD

Offset	Name	Description
0–11	DstNetAddr	Route Table Lookup Value (field value is compared with this)
12–13	DstFrameType	destination frame type (i.e., 802.3, FDDI, etc.)
14–15	DstInterface	destination interface number
16–17	MacHdrLen	length of MAC header and encapsulation to append to frame
18–19	DataLen	length of frame less MAC header and encapsulation
20–21	MinLen	minimum allowable frame size for this destination
22–23	MaxLen	maximum allowable frame size for this destination
24–27	MacHdr	pointer to MAC header and encapsulation to append to frame
28–31	DataPtr	pointer to frame less any bits below the routing protocol header

In Tables 1–11 the records of the protocol description and associated field, statistics, lookup, checksum, vary, route determination and filter records are shown as they appear when resident in memory. In the presently preferred embodiment, each of these protocol description records with its associated field, statistics, lookup, and filter record information is also written to a protocol specific protocol description file (PDF).

In the presently preferred embodiment, the following sequence of data is written to a PDF:

For the Protocol Control Record:

the length of the protocol name including NULL terminator,

the name of the protocol,

the total bit length of the protocol header,

the number of fields required to describe records,

the index of the field currently referenced,.

- the display bit width,
- for each of the field records that describe the protocol header,
  - a call is made to write the field related data (This sequence is described below).
- if the pointer to the option control record is NULL, zero,
- if there are options, the length of the protocol option name including the NULL terminator,
- the option name,
- the option's protocol control record

For the Field Record:

- the flag indicating the value is the actual length of frame,
- the length of the field in bits,
- the byte offset from the start of the protocol header,
- the number of bits to left shift of the 32-bit field,
- the number of bits to right shift of the 32-bit field,
- the number indicating the display type,
- the flag indicating the value is the actual length of the protocol header,
- the reserved byte,
- the multiplier to apply to value prior to display,
- the flag indicating whether to byte swap the 32-bit value,
- the flag indicating that the field is to be displayed,
- the length of the field name including the NULL terminator, or zero
- if the pointer to the statistics structure/class is NULL, zero
- if the pointer to the statistics structure/class is not NULL, a call is made to write the statistics type
- if the pointer to the lookup structure/class is NULL, zero
- if the pointer to the lookup structure/class is not NULL, a call is made to write the lookup type, the number of lookups, and the lookup values

The pointer to vary field, pointer to checksum, pointer to filter, and pointer to route determination are handled similarly.

In the presently preferred embodiment, the initialization of the system includes a determination of the presence of PDF files and the extraction of the protocol and associated control record information from all of the PDF files found. The number of PDF files is determined, and a ProtocolList is constructed consisting of a sorted vector of protocol records at least the size of the number of PDF files found. The name of each protocol record found is inserted in the ProtocolList. The PDF files are then read to memory in the sequence described above for the PDF file writing. The lookup records that indicate a next protocol are associated with the appropriate entries in the ProtocolList.

Two simple protocol descriptions are provided in Tables 12 and 13 (below) to assist in the description of the system of the present invention. The Ethernet Protocol specification shown below is a simplification of an actual Ethernet protocol header.

TABLE 12

ETHERNET v2.0 PROTOCOL SPECIFICATION

0	15	23	47
Destination Hardware Address			
Source Hardware Address			
Ethernet Protocol Type			

Destination Hardware Address	-	destination hardware station address (48 bits)
Source Hardware Address	-	source hardware station address (48 bits)
Ethernet Protocol Type	-	upper layer protocol designator (16 bits) 0x8888=GP

The Ethernet protocol definition described above specifies only one possible upper level protocol, the Generic Protocol (GP) which is indicated by placing a hexadecimal 0×8888 value in the protocol type field. The Generic Protocol (GP) specification shown below in Table 13 has been fabricated to provide examples of different types of field functionalities found in actual network protocols.

TABLE 13

GENERIC PROTOCOL (GP) SPECIFICATION

0	7	15	23	31
Version No.	HeaderLen	Frame Type	Frame Length	
Checksum			Control	Hop Count
Src Socket			Dst Socket	
Src Address				
Dst Address				
0–320 optional field bits				

Version Number	(4 bits) -	software version number
HeaderLen	(4 bits) -	length of GP header in 32 bit words. 0–4 = illegal 5 = No Optional fields, 6–15 = 32–320 bits of options
Frame Type	(8 bits) -	upper level protocol identifier 0 = Unknown 1 = GP1 2 = GP2 3–255 = Unknown
Frame Length	(16 bits) -	frame length in bytes including header
Checksum	(16 bits) -	Checksum of header including options
Control	(8 bits) -	reserved (must be zero)
Hop Count	(8 bits) -	Count of number of networks traversed
Src Socket	(16 bits) -	Socket of Upper-layer protocol sender
Dst Socket	(16 bits) -	Socket of Upper-layer protocol receiver
Src Address	(32 bits) -	Sender protocol address
Dst Address	(32 bits) -	Receiver protocol address

The GP options have two possible formats. Type 1 consists of a single 8-bit option type field containing an option type value. Type 2 contains the 8-bit option type field, but also contains an 8-bit option length field to allow implementation of variable length options in the GP. Two type 1 options and one type 2 option defined in the GP specification are shown below in Table 13:

TABLE 13

End of Option List	(8 bits)	Indicates end of options list. Consists of an 8-bit option type field with value 0. Necessary only for list that does not end on 32-bit boundary.
No Operation	(8 bits)	Performs no function. Consists of an 8-bit option type byte with value 1. Used for alignment of other GP options.
MinMax Size	(32/48 bits)	Allows minimum and maximum allowable frame lengths to be specified. Consists of an 8-bit option type field with value 2, an 8-bit option length field, an 16-bit minimum frame length field, and an optional 16-bit maximum frame length field. If the maximum frame length is specified, the option length field will have value 4, otherwise it will have value 6 specified in units of 8-bit bytes.

Describing the flow charts of FIGS. 11–16 requires the definition of several variables. These variables (described in Table 14 below) are used to implement and monitor the current control logic state of a network interface system in accordance with the present invention:

TABLE 14

FramePtr	Pointer to start of network frame being parsed
HwLen	Bit length of network frame as reported by input device
ParseLen	Number of bits parsed in the current network frame
Current Protocol	Pointer to protocol description control record in use
CurField	Index of field in CurrentProtocol being parsed
ParsePtr	Pointer to start of protocol header in frame being parsed
FrameLen	Number of meaningful bits in the current network frame
ProtoParse Len	Number of bits parsed in the current protocol header
HeaderLen	Size in bits of protocol header being parsed
ParseLvl	Zero based protocol level in ISO reference model of protocol being parsed (current protocol)
ParseList	Array of pointers to protocol headers in frame being parsed (only 0 to (ParseLvl-1) are valid)
SrcIntf	Index of interface on which frame being parsed was received (useful for bridging applications and interface operations)
IntfTypes	Array of values defining the type of each interface in the network system (useful for bridging operations and type specific operations)

Network frames contain one or more protocol headers, an optional number of data bits, and an optional number of pad bits. Frame bits up to the frame length extracted during parsing for which no protocol description exists are considered data. Bits beyond the frame length extracted during parsing are considered to be pad. Two network frames are provided as examples to be used during discussion of the control logic of the present invention:

Frame 1 shown below has a hardware length of eighty-two 8-bit bytes and consists of a fourteen byte Ethernet header, a twenty byte GP header with no option bytes, and forty-eight bytes of application data. No hardware pad is required because the frame length exceeds the Ethernet minimum of sixty bytes.

Frame (1)

(1)	08	00	00	00	00	03	08	00	00	00	00	04	88	88											Ethernet Header	(14)
(2)	35	00	00	44	B1	5F	00	01	08	00	01	47	01	02	03	04	05	06	07	08					GP Header	(20)
(3)	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	Data	(24)
(4)	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	Data	(24)

Frame 2 shown below has a hardware length of sixty 8-bit bytes and consists of a fourteen byte Ethernet header, a twenty-eight byte GP header including eight option bytes, and eighteen bytes of pad to achieve the sixty byte Ethernet minimum frame length requirement.

Frame (2)

(1)	08	00	00	00	00	01	08	00	00	00	00	02	88	88							Ethernet Header	(14)
(2)	37	03	00	2A	FF	FF	00	05	08	00	01	00	01	02	03	04	05	06	07	08	GP Header	(20)
(3)	01																				GP NoOp Option	(1)
(4)	02	04	00	00																	GP MaxSizeOpt	(4)
(5)	00																				GP EOL Options	(1)
(6)	00	00	00																		GP Option Pad	(2)
(7)	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	Frame Pad	(18)

A flow chart is provided for each of the major control logic components of the present invention. The flow chart shown in FIG. **11** outlines ParseFrame control logic in accordance with the present invention and shows how successive protocol headers are parsed, and how remaining information is parsed as application data and frame pad. The flow chart in FIG. **12** outlines ParseProtocol control logic in accordance with one form of the present invention and shows how fixed and optional fields may be parsed in a selected protocol. The flow chart shown in FIG. **13** outlines ParseFields control logic in accordance with the present invention and shows how decisions are made and operations performed on extracted field values. The flow chart shown in FIG. **14** outlines ValidateValue control logic in accordance with the present invention and shows how branching, next protocol determination, and validity decisions are made with extracted field values. The flow chart shown in FIG. outlines ApplyFilter control logic in accordance with one form of the present invention and shows how filter criteria are applied to fields in network frames. The flow chart shown in FIG. **16** outlines GetValue control logic in accordance with one form of the present invention and shows how field values are extracted from network frames. These six components of the control logic of a network interface system in accordance with the present invention are described in detail below.

Referring now to FIG. **13**, a value is extracted by the GetValue control logic (shown in FIG. **16**) of the system (at **210**) for each configured field that is relevant to the current network frame. As shown in FIG. **16**, the fdwoff value is added to ParsePtr to locate and extract a 32-bit value (at **502**) containing the field value, which is then byteswapped (at **510**) if the fswap field is TRUE. If the ptr2vary field is NULL (at **512** or **518**), indicating that the value does not need to be modified or varied, the field value is extracted (at **506**) by shifting the 32-bit value left fshl bits and then right by fshr bits to eliminate any extraneous bits. If the ptr2vary field is non-NULL, the extracted 32-bit value is modified using the configured method (at **514** or **520**) and the resulting 32-bit value overwrites the extracted 32-bit value (at **516** or **522**). If the extracted value was byteswapped (at **510**), the modified value is swapped back (at **516**) prior to overwriting the original value. The extracted value is returned (at **508**) by the GetValue control logic.

It will be appreciated by those skilled in the art that the system of the present invention can handle different machine endian architectures (at **210**) by rearranging the bit and byte order of the extracted 32-bit network frame value to match the target hardware architecture, and can be adapted easily to RISC based architectures where all memory accesses must be aligned in some fashion.

If the GetValue control logic was about to extract a value for the GP HeaderLen field from frame (2), ParsePtr would point at the first value of line 2, from FIG. **5***a*, fdwoff would be 0, fshl would be 4, and fshr would be 28, so that 32-bits of data would be extracted (at **502**) and, possibly, byteswapped (at **510**) to obtain a hexadecimal value equal to 0x3703002A.

In binary notation this is:

0011 0111 0000 0011 0000 0000 0010 1010

Shifting left 4 bits (at **506**) yields:

0111 0000 0011 0000 0000 0010 1010 0000

Shifting right 28 bits (at **506**) yields:

0000 0000 0000 0000 0000 0000 0000 0111

Which in decimal notation is: 7

Therefore, the actual length of the GP header in frame (2) is seven 32-bit words, which is 28 bytes or 224 bits.

Although the presently preferred embodiment of the system of the present invention is designed to handle a maximum field width of 32 bits, it will be appreciated by those skilled in the art that the system may be designed to handle any required maximum field width, and is particularly efficient where the maximum field width matches the underlying hardware architecture. It will also be appreciated by those skilled in the art that it is possible to divide larger protocol fields into sub-fields as demonstrated by the Ethernet protocol field descriptions shown in FIG. **4***a* where the 48-bit hardware address fields have each been defined as two 24-bit sub-fields.

The ValidateValue control logic shown in FIG. **14** is performed on each extracted field value by the ParseFields control logic (at **214**) shown in FIG. **13**. Each field may have an associated lookup structure reference containing one or more values and/or ranges of values that have a particular meaning for that field.

If no lookup structure is configured for a particular field, all values are deemed to be valid (at **318** and **312**), which causes parsing to continue with the next sequentially defined field of the current protocol description.

If a lookup structure exists for a particular field but the extracted value is not found therein (at **314** and **316**), parsing still continues with the next defined field of the current protocol. However, the value is considered invalid.

Values or ranges of values found in configured lookup structures are considered to be valid. The Prot and NextIndex values associated with a value or range found are used to specify NextProtocol, the protocol description (at **308**) to be used after current protocol header parsing is completed, and the index of the next field (at **310**) is used to determine where parsing of the current protocol will continue after the current field. The first valid field parsed in a protocol that specifies the NextProtocol has precedence over all subsequent NextProtocol specifiers (at **306**).

The ValidateValue control logic returns an updated CurField value (at **312** and **316**) together with a valid/invalid indication, and where indicated (at **308**) may return an updated value for NextProtocol.

Using value 0x8888 as an example, if the ValidateValue control logic is applied to the Ethernet Type field and associated lookup structure shown in FIGS. **4***a* and **4***d* respectively, the lookup structure would be found (at **302**), the value will be found in it (at **304**), the associated Protocol field found with the range containing 0x8888 value will be used to update the NextProtocol variable (at **308**) if it is NULL (at **306**), and the associated Next Index field will be used to update the CurField variable.

Using FIG. **5***c* as an example, it may be seen how values may be used to continue parsing at different locations in the current protocol description. In this case, value 0x02 for the Frame Type field causes Checksum field processing to be skipped.

Referring back to the ParseFields control logic shown in FIG. **13**, the ParseFields control logic parses the fields in each protocol header contained in a particular network frame by using field values obtained in accordance with information specified in associated protocol descriptions. The ParseFields control logic is applied for each protocol description required for a particular network frame. If the ParseFields control logic were applied to the exemplary frame, "Frame (1)," described above, the network interface system **10** of the present invention would apply the ParseFields control logic with the protocol descriptions for the Ethernet protocol shown in Table 12, the GP shown in Table 13, and an unspecified Data protocol description.

The ParseFields routine is entered (at **200**) with ParsePtr pointing at the start of a protocol header in a particular network frame and CurrentProtocol set to an appropriate protocol description. Parsing starts at Protocol bit and field zero when CurField and ProtoParseLen are cleared (at **202**), also, HeaderLen is set to the configured protocol control record NumBits value, and LocalProto, the local next protocol return value variable is cleared. Using the Ethernet protocol description shown in FIG. **4** as an example, HeaderLen would be set to 112 bits.

The control loop (at **204** through **224**) continues until the last field has been parsed (at **206**), all bits in the header have been parsed (at **208**), or all bits in the frame have been parsed (at **209**).

For each field a value is retrieved by the system (at **210**). If there is a filter criteria for the field it is applied (at **232**) by the ApplyFilter control logic. The System Filter Status is set to FILTER_FRAME and NextCriteriaIndex is set to zero for every configured filter channel prior to the start of frame processing and after each frame is processed (at **124** in FIG. **11**).

Referring now to the overall system filter channel control structure shown in FIG. **10**, and using the filter expression shown below as an example:

if {(the Ethernet Dst Vendor Address is 0x08FFFF AND the Ethernet Dst Station Address is 0x334455) OR (the GP Frame Type is 1 OR the GP Frame Type is 2)} keep this network frame

we can divide the expression into three distinct filter criteria:

(0) if the Ethernet Dst Vendor Address is 0x08FFFF

(1) if the Ethernet Dst Station Address is 0x334455

(2) if the GP Frame Type is 1 OR the GP Frame Type is 2

FIG. **10**(*a*) shows an example Filter channel structure for the expression shown above and refers to the three Filter Criteria Records of FIG. **10**(*b*) that implement the three filter criteria shown above and refer respectively to FIGS. **10**(*c*), **10**(*d*) and **10**(*e*) which implement the three criteria as lookup structures.

Referring now to FIG. **15**, after the ApplyFilter control logic is entered (at **400**), the Index of one of the filter criteria records shown in FIG. **10**(*b*) is computed with NextCriteriaIndex (at **402** and **404**) for the associated filter channel shown in FIG. **10**(*a*).

If Index is less than NextCriteriaIndex (at **402**) it indicates that this filter criteria does not need to be evaluated. This may occur because a filter channel has been satisfied and NextCriteriaIndex has been set to TotalCriteria to disable further filter processing.

If Index is greater than NextCriteriaIndex (at **404**) this indicates that a filter criteria was skipped in the evaluation of this filter channel which invalidates the filter result. In this case, further filter evaluation is disabled (at **414**) by setting NextCriteriaIndex to TotalCriteria and ApplyFilter returns to the caller.

If Index and NextCriteriaIndex are equal, the field value is found (at **406**) in the associated lookup table, NextCriteriaIndex is updated with the associated NextIndex value and if the associated return value status is PASS_FRAME, the System Filter Status is updated to PASS_FRAME. In this preferred embodiment, the range of possible values for a field must be fully covered. Similarly, in the preferred embodiment a frame will be completely parsed for statistics gathering.

Criteria (0) cannot be used to determine a PASS/FILTER_FRAME result for the filter expression above because it must be logically AND'ed with criteria (1). This is illustrated in FIG. **10***b*, where every value results in no change to the status. The logical AND with criteria (1) is implemented using the NextIndex value. If criteria (0) is FALSE then NextIndex is 2 which causes criteria (1) to be skipped, otherwise NextIndex is 1.

Criteria (1) when TRUE can be used to determine that the filter expression is TRUE because it is not evaluated unless criteria (0) is also TRUE, and the filter expression is the result of ((0) and (1)) or (2). If criteria (2) is FALSE then a PASS/FILTER_FRAME result cannot be determined for the filter expression. This is illustrated by FIG. **10***c*, where the criteria value results in a PASS_FRAME status, and every other value results in no change to the status. The filter expression Count value is reset on completion of frame processing.

Criteria (2) when TRUE can be used to determine that the filter expression is TRUE because it is logically OR'ed with the result of the first two criteria.

It should be noted that the system of the present invention will collect statistics on all fields evaluated regardless of the decision to pass or filter the frame, which may not be acceptable in some instances. It will be appreciated by those skilled in the art that the system of the present invention may be implemented as sequential parsing loops, so that filtering decisions may be made prior to the application of statistics or other field operations.

It will be appreciated by those skilled in the art that the system of the present invention offers significant advantages over traditional filtering methods by allowing filtering criteria to be specified for any subset of bits in any field, by allowing criteria to be applied to every instance of a field that appears more than once in a network frame, and by providing a simple method for easily specifying ranges of values.

Returning again to FIG. **13** after applying a filter criteria, the extracted value is processed by the ValidateValue control logic (at **214**), which updates the NextProtocol and CurField variables and returns a valid/invalid value indication. If ValidateValue returns invalid, parsing of the current field stops (at 216) and restarts with the updated CurField value (at 204), otherwise each configured parsing operation is applied based on the extracted value.

The statistics entity of the field sub-record may be used to indicate categories of statistics to be maintained for each protocol header field (at 218 and 236). Details about mechanisms for collecting statistics are not relevant to the present discussion. However, it will be appreciated by those skilled in the art that the addition of various classes of statistics such as field counters, summing of field contents, and arrays of counters/sums based on field contents may be used in accordance with the present invention. Using FIG. 5*a* as an example, it would be possible to configure the FrameLength field to accumulate an array of counts for each possible field value. From this array, the distribution of GP frames sizes is immediately available, and the length of all GP frames and each frame size may be computed.

Although checksum verification/computation (at 217 and 235) and route determination capabilities (at 219 and 237) are not described in detail in FIG. 13, those skilled in the art will recognize that a system in accordance with the present invention may be configured easily to implement those capabilities. Further, exemplary software listings for implementing these capabilities are provided in the Appendix which is attached hereto and incorporated herein by reference. Moreover, upon review the listings in the Appendix entitled csum.asm, checksum.hpp, route.cpp and route.hpp, those skilled in the art will appreciate that the ability to configure IP, TCP, UDP and IPX checksum capabilities may readily be incorporated into a system in accordance with the present invention. The same is true for a general purpose 16 bit ones complement checksum capability. Finally, those skilled in the art will appreciate that the system of the present invention may be configured in virtually infinite ways to implement virtually any desired checksum capability or, indeed, any desired data manipulation function.

Although in the preferred form the Verify checksum control logic is integrated into the ParseFields control logic (at 217 and 235), the result is not used because processing of frames with bad checksums is device dependent. For example, frames with bad checksums would be counted and saved by a protocol analyzer, while a routing device would count and discard them.

An ability to route frames based on values contained in fields of up to 96 contiguous bits is also demonstrated in the software listings included in the Appendix, and those skilled in the art will recognize that the 96 bit limit may be changed easily to allow for proper handling of protocols with route determination fields longer than 96 bits.

Moreover, those skilled in the art with appreciate that the system of the present invention may be augmented to support virtually any field based operations through modification of the ParseFields control logic loop (at 204–224). For example, it is believed that field based encryption and decryption operations may be added to the system of the present invention with minimal effort.

The HeaderLength field of a protocol description sub-record when non-zero is used to indicate that the extracted value of the current field may be used to compute the length of the current protocol header. The extracted value when multiplied with the configured HeaderLength field value yields the length of the protocol header in the current network frame (at 238). The HeaderLength field is configured to be 32 for the FrameLength field of the GP description shown in FIG. 5*a*. If HeaderLength is used together with the HeaderLen value extracted from frame (2), an actual GP header length of 224 bits (32*7) is calculated. Because the fields defined in FIG. 5*a* add up to only 160 bits, it will then be possible to determine that the (224–160) or 64 bits of optional fields exist in frame (2).

For each field with a valid value, the BitLength field is added to ProtoParseLen, the number of bits parsed in the current protocol, and ParseLen, the number of bits parsed in the network frame (at 222).

The FrameLength field of a protocol description sub-record when non-zero is used to indicate that the extracted value of the current field may be used to compute the length of the current network frame. The extracted value when multiplied with the configured FrameLength value yields the number of meaningful bits in the current frame (at 240). The FrameLength field is configured to be 8 for the FrameLength field of the GP description shown in FIG. 5*a*. If FrameLength is used together with the FrameLen value extracted from frame (1), an actual frame length of 336 bits is calculated (8*42). Because the hardware length of frame (1) is 480 bits (8*60 bytes), it is now possible to determine that the last ((480–366) bits) of frame (2) is pad, added to the frame in this case, to achieve the required Ethernet minimum length of (8*60 bytes). In a preferred form, the length computed for the frame is verified against the length provided by the hardware, and the minimum of the two values is be used as FrameLen.

If every field in the current protocol has been parsed (at 206), or every bit in the current protocol header has been parsed (at 208), or every bit in the current frame has been parsed (at 209), parsing of the current protocol terminates. If LocalProto is NULL (at 225) when parsing of the current protocol terminates, ParseProtoLen is added to ParsePtr (at 228) so that it points to the start of the next protocol header in the frame. If LocalProto is not NULL (at 225) when parsing of the current protocol terminates and there are unparsed header bits remaining (at 226), ParseLen and ProtoParseLen are adjusted to account for each unparsed header bit (at 227) before adding ProtoParseLen to ParsePtr (at 228). In every case, ParseFields control logic returns LocalProto (at 230).

Referring now to FIG. 11, the ParseFrame control logic of the present invention, network frames are composed of one or more protocol headers which in turn are composed of one or more predefined contiguous bit fields. The ParseFrame control logic systematically parses through each network frame (at 104 to 108 and 128) until all known protocol headers have been parsed. Any remaining frame bits are parsed as application data (at 110, 112 and 130) and/or pad data (at 114, 116 and 132).

Referring now to FIG. 12, ParseProtocol control logic where all fixed and optional protocol fields are parsed is entered (at 200) with ParsePtr and ParseLen initialized from prior processing. All protocol fields that are fixed in location are parsed (at 152). If all bits in the frame are parsed (at 154) after parsing fixed fields, frame parsing is complete and ParseProtocol returns NULL (at 168). If there are more bits in the frame to parse and the current protocol description supports optional fields (at 156) and the current frame contains optional fields (at 160) they are parsed (at 160 to 166) using the current protocol option control protocol description as a starting point (at 158). Once all options are parsed (at 172) ParseProtocol will return NULL (at 168) if all bits in the frame have been parsed or will return LocalProto (at 170) if more bits remain to be parsed.

Referring again to FIG. 11, once the system has received a network frame (at 100), defined by an interface number (SrcIntf), a frame location (FramePtr) and a hardware length (HwLen), the frame is resolved into its protocol and field components using the system of the present invention.

Using the exemplary frame, "Frame (2)," described above as an example, the system (at 102) in FIG. 11 would obtain from the receiving network interface device SrcIntf, the receiving interface number, FramePtr, a pointer to the frame, and HwLen, the hardware frame length. The hardware length of frame (2) is 480 bits. ParseLen, the number of bits in the frame that have been parsed, ParseLvl and CurField, the index of the protocol field being processed are reset to zero, and CurrentProtocol, is set up with the initial protocol description structure of the receiving interface number which for frame (2) is the Ethernet Protocol description defined in FIGS. 4–4*d*. FrameLen is set to the value of HwLen, and ParsePtr is set to the value of FramePtr.

Each field in the Ethernet Protocol description as shown in FIG. 4*a* is parsed (at 106) using the ParseProtocol control logic shown in FIG. 13.

The ParseProtocol control logic updates ProtoParseLen, the number of bits parsed in the CurrentProtocol, HeaderLen, the protocol header size determined during parsing, and returns NextProtocol, a reference to the next applicable protocol description structure to use during parsing. ParseProtocol also updates ParsePtr and ParseLen by adding ProtoParseLen to them. If NextProtocol is NULL, the remaining frame bits will be treated as Data and/or Pad bits.

After the Ethernet protocol fields in frame (2) are parsed (at 106) by the ParseProtocol control logic shown in FIG. 13, HeaderLen, ParseLen and ProtoParseLen will be 112 bits, NextProtocol will refer to the GP shown in FIGS. 5–5(*e*), and ParsePtr will point at the start of line 2 in frame (2). CurrentProtocol will be updated with the NextProtocol value of GP (at 130) and the GP fields in frame (2) are parsed (at 106) by the ParseFields control logic shown in FIG. 13, which will update HeaderLen and ProtoParseLen to be 160 bits, and return NextProtocol as NULL. ParsePtr will point at the start of line 3 in frame (2), and ParseLen will be updated to 272 bits.

Referring now to FIG. 12, if a CurrentProtocol such as GP shown in FIGS. 5–5*e* supports optional fields, which is indicated by the Options field of the control record, then any options in the network frame are sequentially parsed (at 160–166) using the ParseFields control logic (at 164) until ProtoParseLen, the number of bits parsed in the protocol header is equal to HeaderLen, the protocol header size determined during parsing with the original protocol description (at 152).

Using the exemplary frame, "Frame (2)," described above as an example, after the GP fields are parsed (at 152), HeaderLen will be updated to 224 bits, while ProtoParseLen will be updated to 160 bits, which indicates the presence of (224–160) bits of optional fields (at 160).

Every protocol description with optional fields will have a master protocol option description, and one option protocol description for each supported option. Using the GP protocol control record shown in FIG. 5 as an example of how optional fields might be described, the Options field will refer to a master option control record similar to FIG. 6. The master option control record will contain one field (see FIG. 6*a*) that does not contribute to the number of bits parsed (BitLength zero) with an associated lookup structure (see FIG. 6*b*) for each possible option. Using FIG. 6*b* as an example, each defined option refers to an appropriate protocol option description. The first field of each option description has an associated lookup structure (see FIGS. 7*b*, 8*b*, and 9*b*) that refers back to the master option control record. FIG. 9*a* shows how optional fields with variable length may be handled by computing the frame length.

Referring now to FIG. 13, in a preferred form the Option Type field in a frame is examined twice, once with the master protocol option description and once with the appropriate option protocol description. If an unknown option is encountered (any value between 0x03 and 0xff inclusive for FIG. 6*b*), ParseLen, ProtoParseLen, and ParsePtr are updated (at 227 in FIG. 13) to skip any remaining options and parsing of the frame continues with the LocalProto protocol description returned (at 230).

Referring again to FIG. 12, and using the GP control record shown in FIG. 4 as an example, the system would determine (at 156) that the CurrentProtocol supports options and (at 158) will update CurrentProtocol to the master option descriptor of FIG. 6. The master option control record has one field shown in FIG. 6*a*, which is used to select the appropriate option protocol description structure to use. The lookup structure shown in FIG. 6*b* allows option descriptions to be associated with option type values extracted from network frames. The system (at 160–166) will parse one option at a time in the current network frame until all options are parsed.

Before each option is parsed, the number of bits parsed using the previous option protocol control record is subtracted from HeaderLen (at 162). The ParseFields control logic is alternately processed (at 164) with the master protocol option control record, and an appropriate option control record. The CurrentProtocol is updated (at 166) with the NextOption value returned by ParseFields, and the loop is re-entered (at 160).

Using the exemplary frame, "Frame (2)," described above as an example with ParsePtr pointing at line 3, and CurrentProtocol pointing at the GP master option description shown in FIG. 6, it may be seen how a NumBits value of zero prevents the master option description from contributing to the number of bits parsed (at 162), and how ParseFields and ValidateValue use the master option description field to select an appropriate GP option description structure from the lookup structure of FIG. 6*b*. For frame (2), the first option byte at line 3 contains value 1, which causes the GP NoOp option description structure shown in FIG. 8 to be selected (at 166). The NoOp NumBits value of 8 is added to ProtoParseLen (at 162), and the single field defined in FIG. 8*a* is parsed at 164. In a preferred form, each option description structure must have a field with an associated lookup structure that is always processed and refers back to the master option description structure.

Thus, for "frame (2)" the option processing control loop (at 160 through 166) is alternately applied with the description structures of FIG. 6 and FIGS. 8, 9, and 7. The GP End Of List Option does not refer back to the master option description because it indicates an end to option processing. Any remaining option bits are assumed to be pad and are disregarded so that the check for more options (at 160 in FIG. 12) will fail and return to frame protocol parsing (at 108 in FIG. 11).

Once all options have been parsed in frame (2) or the system (at 160) determines that the current frame has no optional fields as in frame (1), the system control logic (at 168 or 170) will return to the main parsing loop (at 108 in FIG. 11).

While the invention of this application is susceptible to various modifications and alternative forms, specific examples thereof have been shown in the drawings and are herein described in detail. It should be understood, however, that the invention is not to be limited to the particular forms or methods disclosed, but to the contrary, the invention is to cover all modifications, equivalents, and alternatives falling within the spirit and scope of the appended claims.

What is claimed is:

1. A network filtering system comprising:

means for storing in a first memory a plurality of programmably configurable protocol descriptions, said plurality of programmably configurable protocol descriptions defining one or more filter criteria;

means for storing in a second memory a program for controlling a data filtering function to be executed by a processing unit, said program including instructions for causing said processing unit to selectively retrieve at least one of said plurality of programmably configurable protocol descriptions from said first memory and to vary the execution of said data filtering function based upon said at least one retrieved protocol description;

means for delivering to said processing unit said program for controlling said data filtering function;

means for enabling said processing unit to execute said data filtering function; and

means for delivering to said processing unit said data transmitted over a data communications network.

* * * * *

US006141686A

United States Patent [19]

Jackowski et al.

[11] **Patent Number: 6,141,686**

[45] **Date of Patent: Oct. 31, 2000**

[54] **CLIENT-SIDE APPLICATION-CLASSIFIER GATHERING NETWORK-TRAFFIC STATISTICS AND APPLICATION AND USER NAMES USING EXTENSIBLE-SERVICE PROVIDER PLUGIN FOR POLICY-BASED NETWORK CONTROL**

[75] Inventors: **Steven J. Jackowski**, Santa Cruz; **Christopher N. Thomas**, Soquel, both of Calif.

[73] Assignee: **Deterministic Networks, Inc.**, Santa Cruz, Calif.

[21] Appl. No.: **09/103,339**

[22] Filed: **Jun. 23, 1998**

Related U.S. Application Data

[63] Continuation-in-part of application No. 09/042,306, Mar. 13, 1998.

[51] **Int. Cl.**^{7} **G06F 13/38**; G06F 15/17

[52] **U.S. Cl.** **709/224**; 709/105; 709/229; 709/206; 709/301; 709/302

[58] **Field of Search** 709/224, 226, 709/229, 104, 105, 301, 302, 304, 238, 206, 207, 103, 249; 710/10; 370/252, 254

[56] **References Cited**

U.S. PATENT DOCUMENTS

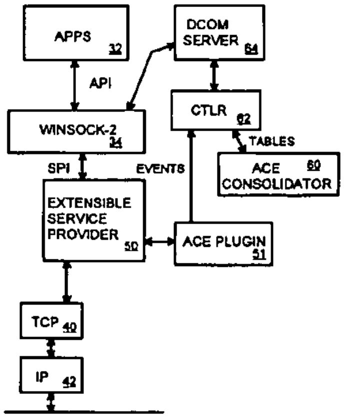
5,136,581	8/1992	Meuhrcke	370/62
5,341,477	8/1994	Pitkin et al.	395/200
5,499,343	3/1996	Pettus	395/200.2
5,596,720	1/1997	Hamada et al.	395/200
5,621,734	4/1997	Mann et al.	370/94.1
5,644,715	7/1997	Baugher	395/200.2
5,668,998	9/1997	Mason et al.	395/701
5,673,322	9/1997	Pepe et al.	380/49
5,674,003	10/1997	Andersen et al.	364/514
5,682,478	10/1997	Watson et al.	395/200
5,682,482	10/1997	Burt et al.	395/242
5,694,548	12/1997	Baugher et al.	395/200.1
5,701,484	12/1997	Artsy	395/683
5,732,270	3/1998	Foody et al.	395/683
5,802,320	9/1998	Baehr et al.	709/249
5,867,667	2/1999	Butman et al.	709/249
5,880,740	3/1999	Halliday et al.	345/435
5,920,705	7/1999	Lyon et al.	395/200.7
5,940,390	8/1999	Berl et al.	370/389
5,943,496	8/1999	Li et al.	395/685
5,956,721	9/1999	Douceur et al.	707/10
5,960,177	9/1999	Tanno	395/229
5,966,531	10/1999	Skeen et al.	395/683
5,974,549	10/1999	Golan	713/200
5,983,274	11/1999	Hyder et al.	709/230
5,987,611	11/1999	Freund	713/201
5,991,302	11/1999	Berl et al.	370/400
5,991,760	11/1999	Gauvin et al.	707/10
6,014,700	1/2000	Bainbridge et al.	709/226
6,034,964	3/2000	Fukushima et al.	370/401
6,076,168	6/2000	Fiveash et al.	713/201
6,078,953	6/2000	Vaid et al.	709/223

Primary Examiner—Le Hien Luu
Assistant Examiner—Bunjob Jaroenchonwanit
Attorney, Agent, or Firm—Stuart T. Auvinen

[57] **ABSTRACT**

Low-level network services are provided by network-service-provider plugins. These plugins are controlled by an extensible service provider that is layered above the TCP or other protocol layer but below the Winsock-2 library and API. Policy servers determine priority of network traffic through control points on a network. Examining packets passing through these control points provides limited data such as the source and destination IP address and TCP ports. Many applications on a client machine may use the same IP address and TCP ports, so packet examination is ineffective for prioritizing data from different applications on one client machine. Often some applications such as videoconferencing or data-entry for corporate sales are more important than other applications such as web browsing. A application-classifier plugin to the extensible service provider intercepts network traffic at above the client's TCP/IP stack and associates applications and users with network packets. These associations and statistics such as maximum, average, and instantaneous data rates and start and stop time are consolidated into tables. The policy server can query these tables to find which application is generating network traffic and prioritize the traffic based on the high-level application. Bandwidth-hogging applications such as browsers can be identified from the statistics and given lower priority.

20 Claims, 14 Drawing Sheets



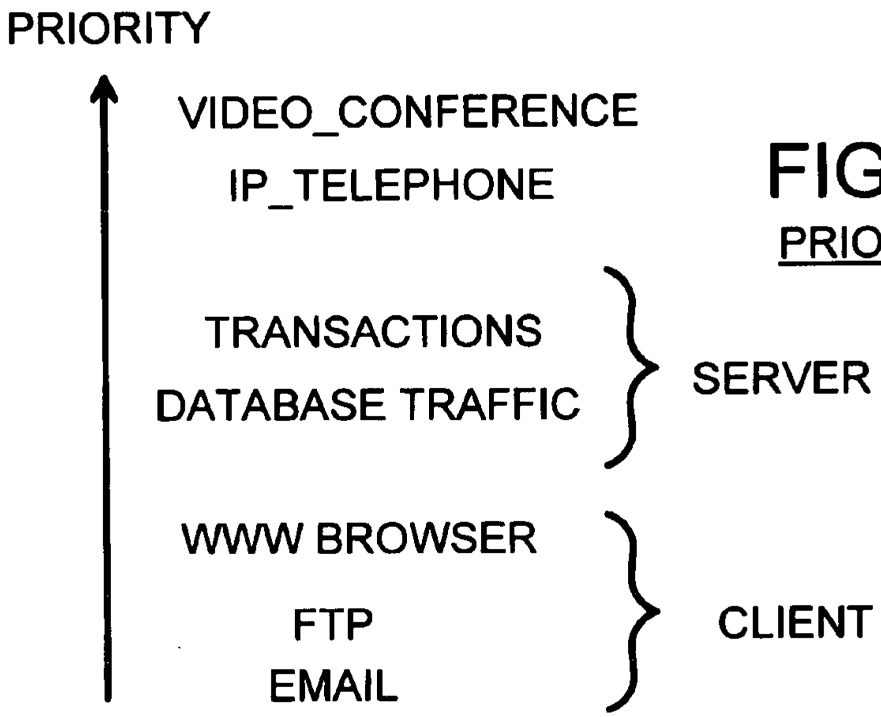
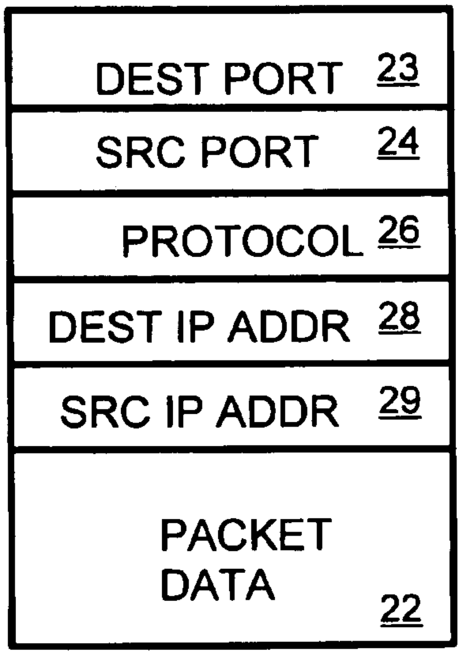


FIG. 1
PRIOR ART



PRIOR ART

FIG. 2

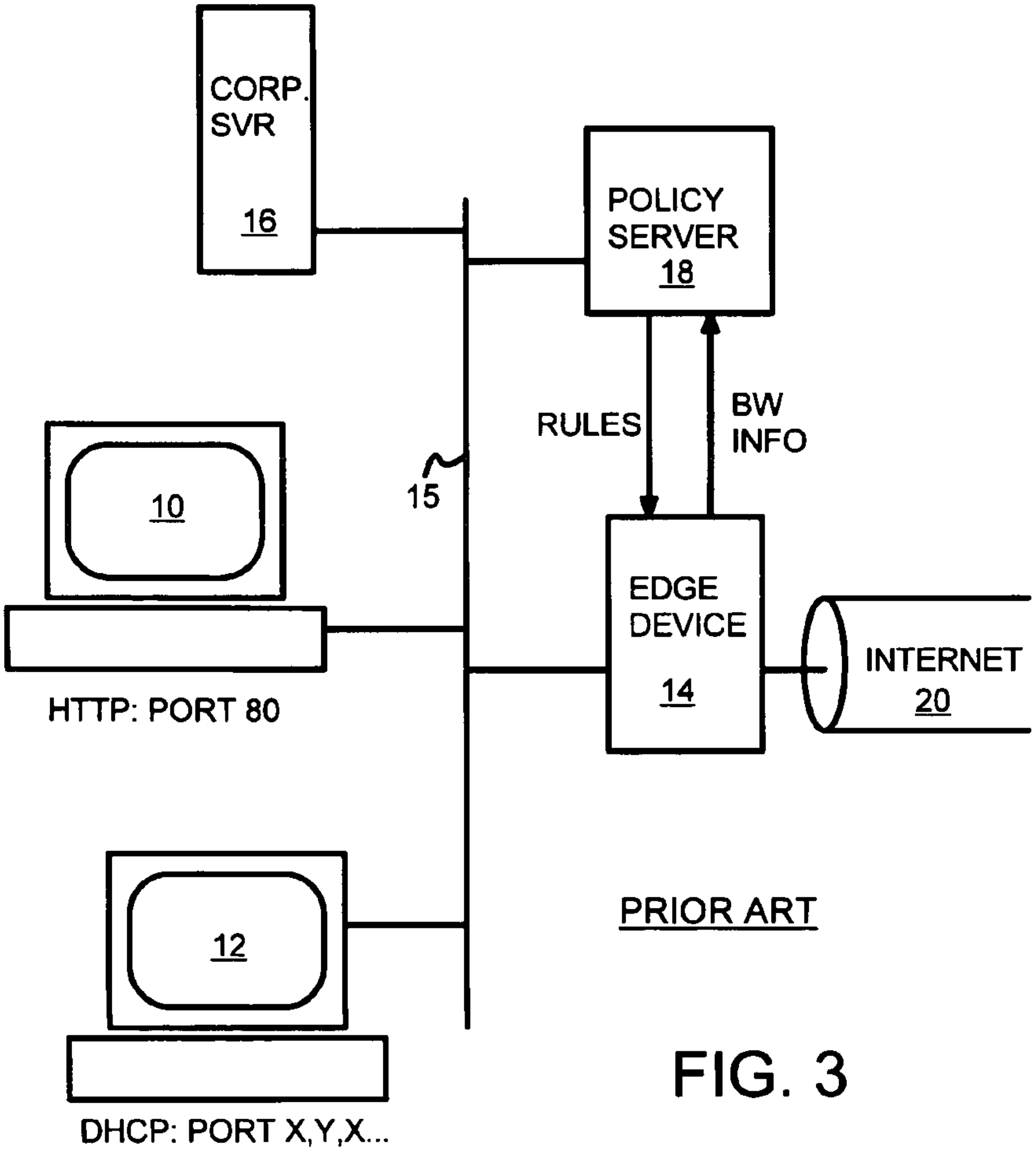

CORP. SVR
16
POLICY SERVER
18
RULES
BW INFO
15
10
HTTP: PORT 80
EDGE DEVICE
14
INTERNET
20
12
DHCP: PORT X,Y,X...
PRIOR ART

FIG. 3

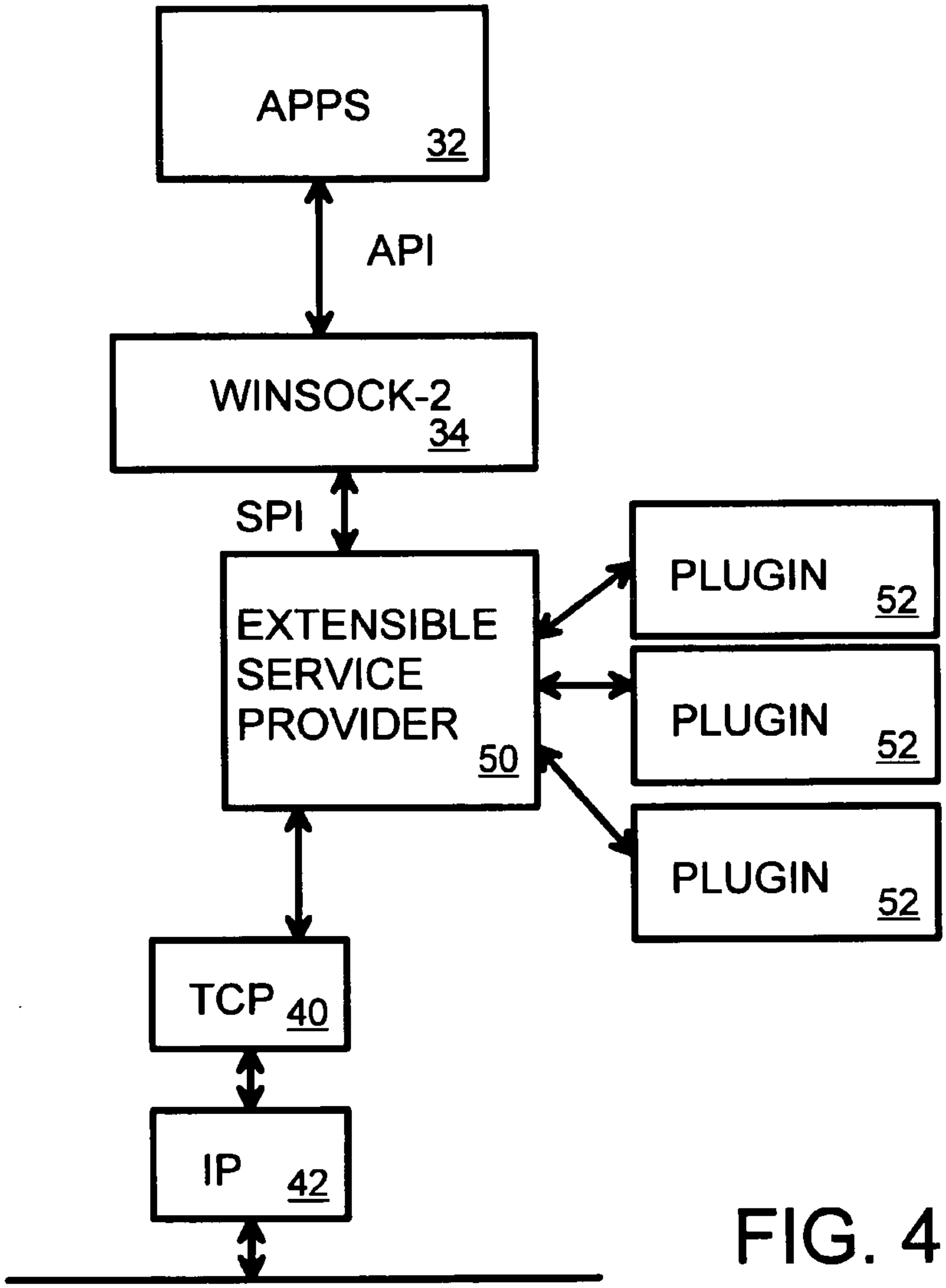

APPS
32
API
WINSOCK-2
34
SPI
EXTENSIBLE SERVICE PROVIDER
50
PLUGIN
52
PLUGIN
52
PLUGIN
52
TCP
40
IP
42

FIG. 4

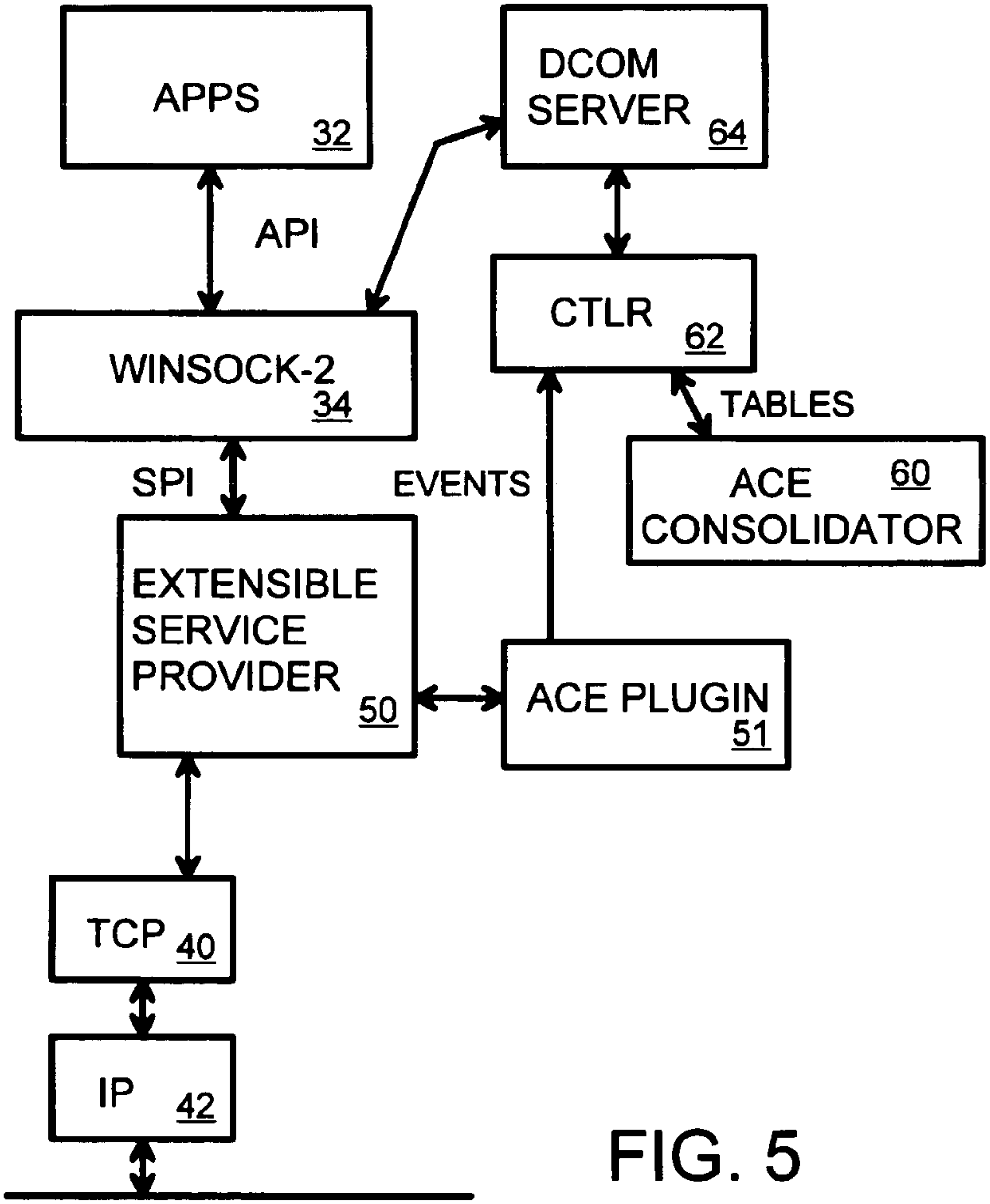

APPS 32
DCOM SERVER 64
API
CTLR 62
WINSOCK-2 34
TABLES
SPI
EVENTS
ACE CONSOLIDATOR 60
EXTENSIBLE SERVICE PROVIDER 50
ACE PLUGIN 51
TCP 40
IP 42

FIG. 5

	TRIGGERING EVENT	OBJECT TYPE	OBJECT NAME
APP	STARTUP CLEANUP	EventAppInit	APPSTART APPSTOP
SOCKET	SOCKET_OPEN SOCKET_CLOSE	EventSocketInit	SOCKETOPEN SOCKETCLOSE
	BIND_COMPLETE ACCEPT_COMPLETE CONNECT_COMPLETE	EventSocketState	SOCKETBIND SOCKETACCEPT SOCKETCONNECT
	RECV_COMPLETE SEND_COMPLETE	EventSocketDataXfer	SOCKETIN SOCKETOUT
FLOW	CONNECT_COMPLETE SEND_COMPLETE SOCKET_CLOSE RECV_COMPLETE	EventFlowInit	FLOWSTART FLOWSTART FLOWSTOP

FIG. 6

Application Open/Close Event

Index	C Type (VB)	Value	Description
0	long (Long)	Version	Version
1	long (Long)	Time	The time of the event. This is the C time_t value representing the number of seconds since 1/1/70
2	long (Long)	ProcessID	The process ID of the application
3	BSTR (String)	Application Name	The name of the application
4	BSTR (String)	User Name	The name of the currently logged in user.

Figure 7A

Socket Open/Close Event

Index	C Type (VB)	Value	Description
0	long (Long)	Version	Version
1	long (Long)	Time	The time of the event. This is the C time_t value representing the number of seconds since 1/1/70
2	long (Long)	ProcessID	The process ID of the application
3	long (Long)	Socket	The socket ID
4	long (Long)	Flow ID	The flow ID of the flow associated with this socket
5	long (Long)	Network Protocol	The network protocol (IP,IPX) associated with this socket.
6	long (Long)	Transport Protocol	The transport protocol used by the socket. These values may be specific to particular network protocols.

Figure 7B

Socket Bind/Connect/Accept Event

Index	C Type (VB)	Value	Description
0	long (Long)	Version	Version
1	long (Long)	Time	The time of the event. This is the C time_t value representing the number of seconds since 1/1/70
2	long (Long)	ProcessID	The process ID of the application
3	long (Long)	Socket	The socket ID
4	long (Long)	Flow ID	The flow ID of the flow associated with this socket
5	long (Long)	Local Address	The IP address and port information
6	long (Long)	Local Port	
7	long (Long)	Remote Address	The IP address and port information
8	long (Long)	Remote Port	

Figure 7C

Flow Table

Index	C Type (VB)	Value	Description
0	long (Long)	Version	Version
1	long (Long)	Number of Flows	The number of flows in the table
2	SAFEARRAY of VARIANT	Flows	This is an array of flows, where each flow is a Variant as described in the Figure 9A.

Figure 9B

Socket Data In/Out Event

Index	C Type (VB)	Value	Description
0	long (Long)	Version	Version
1	long (Long)	Time	The time of the event. This is the C time_t value representing the number of seconds since 1/1/70
2	long (Long)	ProcessID	The process ID of the application
3	long (Long)	Socket	The socket ID
4	long (Long)	Flow ID	The flow ID of the flow associated with this socket
5	long (Long)	Local Address	The IP address and port information
6	long (Long)	Local Port	
7	long (Long)	Remote Address	The IP address and port information
8	long (Long)	Remote Port	
9	long (Long)	Bytes	The number of bytes sent or received

Figure 7D

Flow Start/Stop Event

Index	C Type (VB)	Value	Description
0	long (Long)	Version	Version
1	long (Long)	Time	The time of the event. This is the C time_t value representing the number of seconds since 1/1/70
2	long (Long)	ProcessID	The process ID of the application
3	long (Long)	Flow ID	The flow ID of the flow associated with this socket
4	BSTR (String)	Application Name	The name of the application
5	BSTR (String)	User Name	The name of the currently logged in user.
6	long (Long)	Network Protocol	The network protocol (IP,IPX) associated with this socket.
7	long (Long)	Transport Protocol	The transport protocol used by the socket. These values may be specific to particular network protocols.
8	long (Long)	Local Address	The IP address and port information
9	long (Long)	Local Port	
10	long (Long)	Remote Address	The IP address and port information
11	long (Long)	Remote Port	

Figure 7E

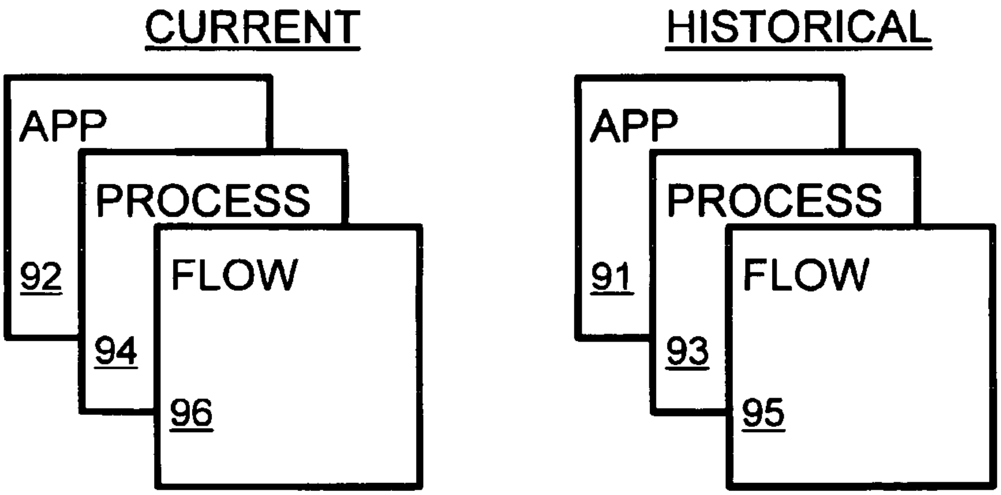

CURRENT
APP
92
PROCESS
94
FLOW
96
HISTORICAL
APP
91
PROCESS
93
FLOW
95

FIG. 8

Flow Information

Index	C Type (VB)	Value	Description
0	long (Long)	Version	Version
1	long (Long)	FlowID	The flow to which the information relates.
2	long (Long)	ProcessID	The process ID of application generating this flow.
3	BSTR (String)	Application Name	The name of the application generating this flow
4	BSTR (String)	User Name	The name of the currently logged in user.
5	long (Long)	Start time	The time the flow began.
6	long (Long)	Stop time	The time flow ended. Zero for ongoing flow.
7	long (Long)	Transport Protocol	Transport protocol.
8	long (Long)	Net Protocol	Protocol
9	long (Long)	Source Port	These entries identify the IP address information for the flow.
10	long (Long)	Source Address	
11	long (Long)	Destination Port	
12	long (Long)	Destination Address	
13	long (Long)	Total Sent	Total bytes sent in this flow
14	long (Long)	Total Received	Total bytes received in flow
15	long (Long)	Max Rate	These are the rates for the life of the flow for bytes sent and received.
16	long (Long)	Min Rate	
17	long (Long)	Average Rate	
18	long (Long)	Delta Sent	Total Sent, Total Received and Average Rate, but reset for each query. Delta Rate is calculated over the time between the queries.
19	long (Long)	Delta Received	
20	long (Long)	Delta Rate	

Figure 9A

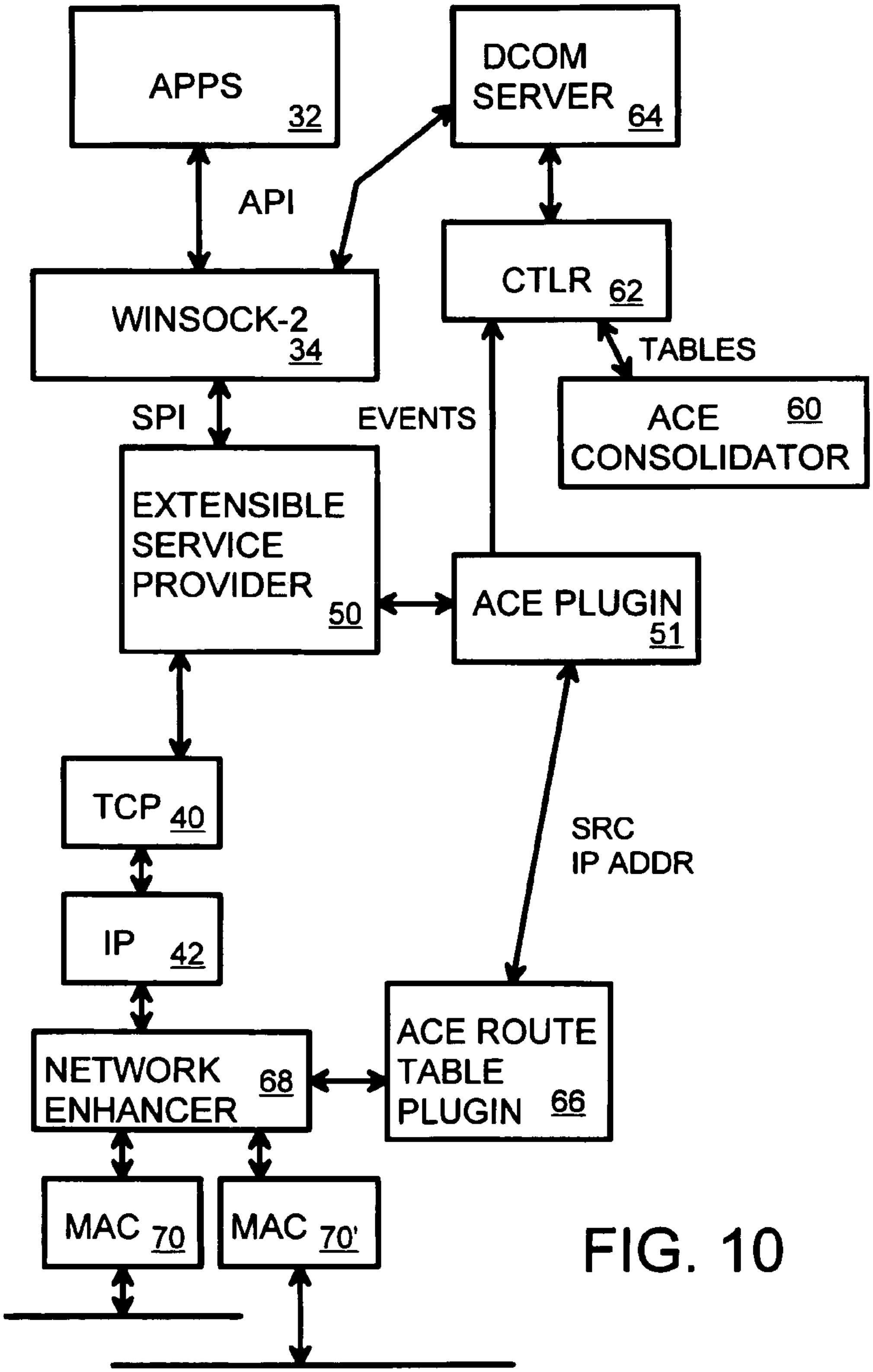

APPS
32
API
DCOM
SERVER
64
WINSOCK-2
34
CTLR
62
TABLES
SPI
EVENTS
ACE
60
CONSOLIDATOR
EXTENSIBLE
SERVICE
PROVIDER
50
ACE PLUGIN
51
TCP
40
SRC
IP ADDR
IP
42
NETWORK
ENHANCER
68
ACE ROUTE
TABLE
PLUGIN
66
MAC
70
MAC
70'

FIG. 10

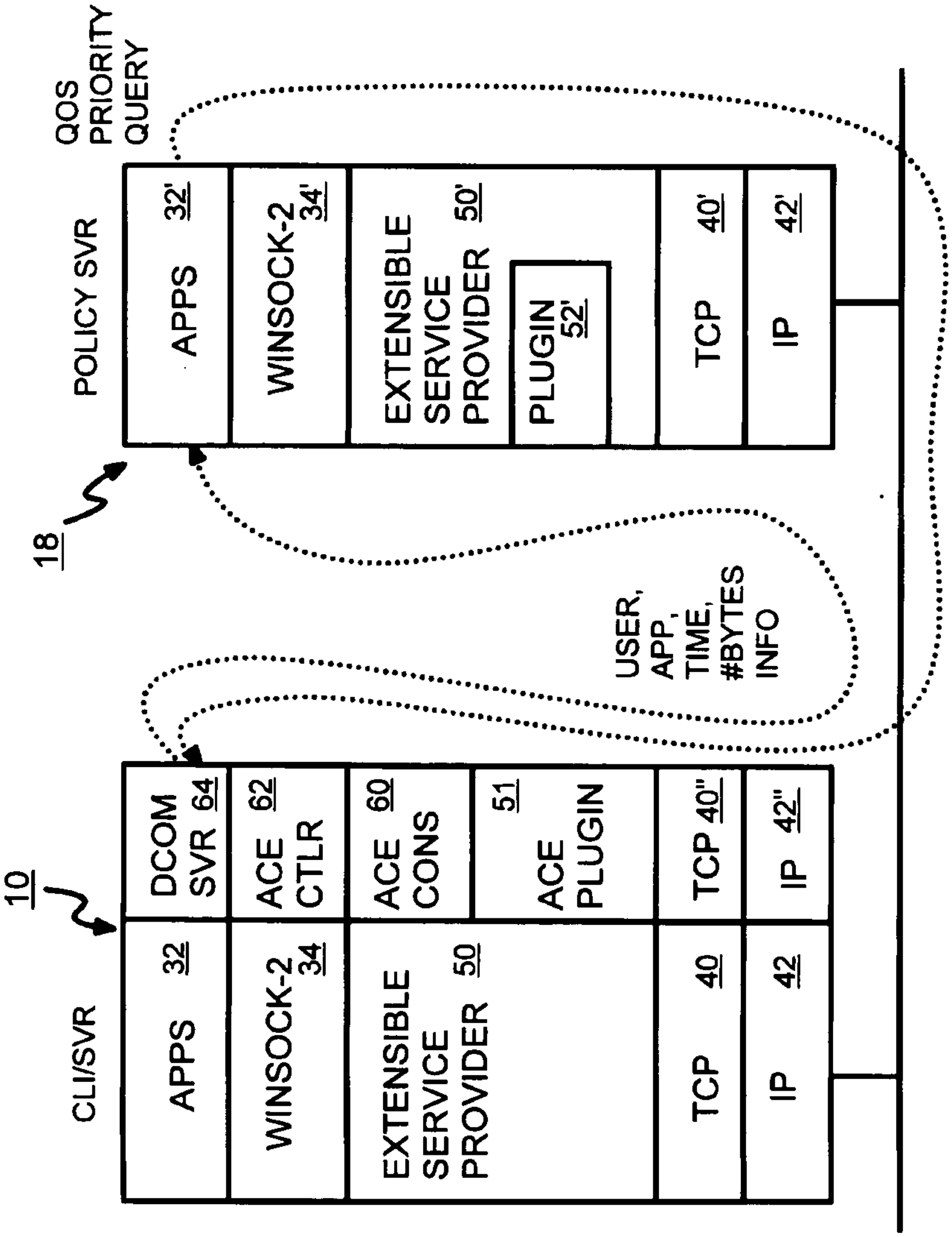

QOS
PRIORITY
QUERY
POLICY SVR
APPS 32'
WINSOCK-2 34'
EXTENSIBLE
SERVICE
PROVIDER 50'
PLUGIN
52'
TCP 40'
IP 42'
18
USER,
APP,
TIME,
#BYTES
INFO
10
CLI/SVR
APPS 32
WINSOCK-2 34
EXTENSIBLE
SERVICE
PROVIDER 50
TCP 40
IP 42
DCOM
SVR 64
ACE
CTLR 62
ACE
CONS 60
51
ACE
PLUGIN
TCP 40"
IP 42"

FIG. 11

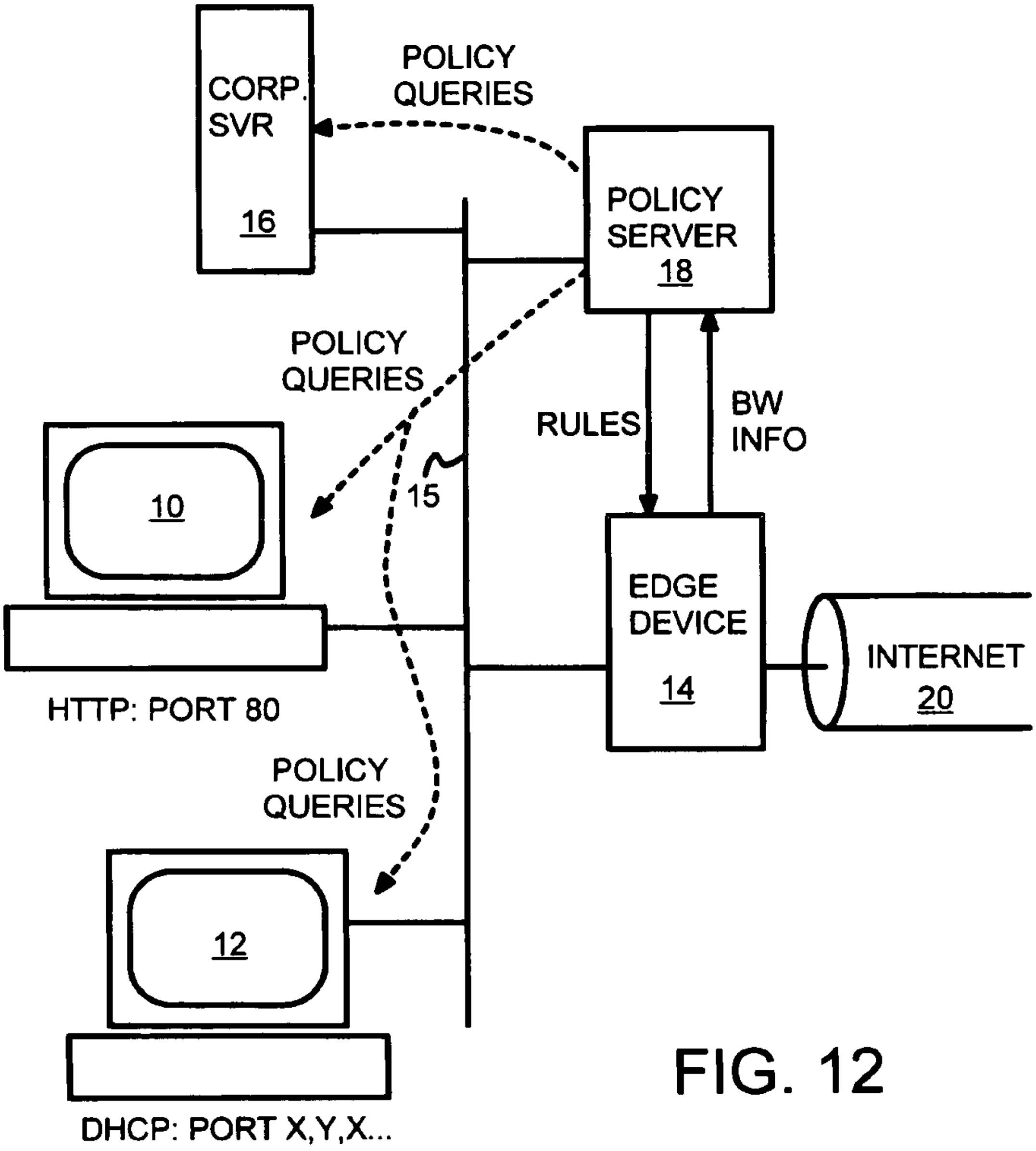

CORP.
SVR
16
POLICY
QUERIES
POLICY
SERVER
18
POLICY
QUERIES
RULES
BW
INFO
10
15
EDGE
DEVICE
14
INTERNET
20
HTTP: PORT 80
POLICY
QUERIES
12
DHCP: PORT X,Y,X...

FIG. 12

CLIENT-SIDE APPLICATION-CLASSIFIER GATHERING NETWORK-TRAFFIC STATISTICS AND APPLICATION AND USER NAMES USING EXTENSIBLE-SERVICE PROVIDER PLUGIN FOR POLICY-BASED NETWORK CONTROL

RELATED APPLICATION

This application is a continuation-in-part of the co-pending application for "Ordering of Multiple Plugin Applications Using Extensible Layered Service Provider with Network Traffic Fltering", U.S. Ser. No. 09/042,306, filed Mar. 13, 1998, now pending.

FIELD OF THE INVENTION

This invention relates to software for computer networks, and more particularly to client plugins for identifying application traffic-signatures, signaling traffic priorities, and for generating network statistics for policy servers.

BACKGROUND OF THE INVENTION

Computer networks such as the Internet are driving increased acceptance of personal computers. Network communication began with text-only e-mail and file-transfer protocols (FTP), but with improved user interfaces and graphics, graphical browsing has become commonplace. Mission-critical business transactions, corporate database queries, and even video conferencing and voice telephone calls all use the Internet.

Not surprisingly, the Internet and local networks are becoming crowded. Simply increasing bandwidth is expensive and often only shifts bottlenecks to another part of the network. While users may not notice delayed e-mail, Internet browsing can become painfully slow during times of network congestion. Video conferencing and telephony suffer poor quality and even gaps of lost speech when the network is slow.

FIG. 1 illustrates differing priorities of various kinds of network traffic. Two-way video and audio communications such as video conferencing and Internet telephony must have their packets delivered over the network in real time, or parts of the conversation are lost. Thus these services must have the highest priority in most networks. Business-critical applications such as financial transactions and accesses of corporate databases have moderately high priority. Browser traffic to the world-wide-web has a lower priority since much of this traffic is for information gathering and personal uses. However, browser traffic should not be so slow as to irritate the users. Lowest in priority are file transfers and e-mail, since these are usually not needed immediately.

Server traffic tends to have a higher priority than client traffic, since business-critical applications reside on corporate servers. Clients are usually individual desktop PC's.

Quality-Of-Service Network Policy

Attempts have been made to improve transmission speed of higher-priority traffic. Bandwidth-shaping or traffic-shaping delays low-priority traffic so that higher-priority packets can pass through with less delay. Quality-of-Service (QOS) is thus improved. Bandwidth can be reserved for the highest-priority applications such as video conferencing. See for example, U.S. Pat. Nos. 5,644,715 and 5,694,548, by Baugher et al., assigned to IBM; also U.S. Pat. No. 5,673,322 by Pepe et al., and U.S. Pat. No. 5,136,581 by Muehrchke, assigned to Bell Labs.

Can't Determine Priority of IP Packets—FIG. 2

Ideally, a network device such as a router, would read a packet's header and determine the priority of that packet from fields in the header. Unfortunately, determining the priority of packets passing through a network point is problematic. Simple filtering software can be used to identify packets using certain network protocols such as TCP, or certain Internet Protocol (IP) addresses.

FIG. 2 shows an Internet packet. Port fields **23**, **24** identify which ports were assigned by the network software for communication with a higher-level application requesting a communications session. Destination port field **23** specifies the port on the destination machine, while source port field **24** specifies the port on the source machine. Protocol **26** is a field identifying the network protocol used, such as TCP or UDP. Destination address field **28** contains the IP address that the packet is being sent to, while source address field **29** contains the IP address of the sender of the packet. The contents or data of the packet, perhaps with additional higher-level headers, is contained in data field **22**.

While some applications may use certain ports, many applications use standard ports, such as port **80** for web browsers. Sometimes these ports are dynamically assigned to applications, so that different ports are used by the same application at different times. Simply reading port fields **23**, **24** does not uniquely identify applications, so it is difficult to determine priority based on port fields **23**, **24**. Most applications use the TCP protocol, so protocol field **26** likewise does not uniquely identify users or their applications.

IP address fields **28**, **29** often uniquely identify a user or a server machine, and IP-address filtering has been used to restrict access by children to adult-only web sites. IP-address filtering has been less successful for blocking unwanted "junk" or "Spam" e-mail, since the IP-address fields are often altered to hide the originating IP address. Larger web sites may use many IP addresses that may dynamically change as the web site is updated. Even client machines can have dynamically-assigned IP addresses rather than a static IP address. In some organizations, many users share an IP address. Thus determining packet priority using IP addresses is not effective.

Ideally, the names of the high-level application on the client and on the server machines should be collected. The high-level application names could then be uses for prioritizing IP packets from these applications, rather than use IP addresses and TCP ports.

Policy-Controlled Network—FIG. 3

FIG. 3 is a diagram of a network that controls traffic using policy rules. Client-PCs **10**, **12** send IP packets over local network **15** to corporate server **16** and Internet **20**. Edge device **14** is a router, switch, gateway, modem or other network device that connects local network **15** to Internet **20**. Traditionally, routers such as edge device **14** have simply passed all packets through roughly in the order received, without regard to priority.

Edge device **14** is able to block or delay packets to and from Internet **20** so that higher-priority packets experience less delay than lower-priority packets. Edge device **14** may examine packets and apply policy rules to determine which packets to accelerate and which to delay.

Policy server **18** sends the policy rules to edge device **14**. Bandwidth information is sent back from edge device **14** to policy server **18**. This bandwidth information might indicate the current bandwidth available to Internet 20 or local network 15, or other traffic or load statistics such as the kinds of packets appearing. The bandwidth information may be used by policy server 18 to re-prioritize packets passing through edge device 14 by adjusting the policy rules sent to edge device 14. For example, when edge device 14 detects video conferencing packets passing through, policy server 18 can reduce the bandwidth allocated to other kinds of packets to reserve additional network bandwidth for video-conferencing packets.

Often higher-priority packets are generated by corporate server 16 than client PCs 10, 12. Policy server 18 can set a higher-priority policy for corporate server 16 when certain kinds of packets appear in the bandwidth information from edge device 14.

While such a policy-controlled network is effective, newer technologies make determining the priority of packets more difficult. Low-priority web browsing from client 10 can be identified by the IP address for client 10 and port 80 used by the browser. However, newer software installed on client PC 12 dynamically assigns ports to applications. IP addresses may also be changed using Dynamic Host Configuration Protocol (DHCP). The application may appear as port 50 one day, but port 22 on another day. The IP address assigned to client PC 12 may also be dynamically assigned or even shared by other client PCs.

Identifying web browser traffic from client PC 12 is thus quite difficult. Client PC 12 could be downloading huge graphics images from the Hubbell Space Telescope for personal use, swamping the capacity of the network, while client 10 waits to read text-based information from an important customer over Internet 20. Network chaos erupts when even a few users hog bandwidth for low-priority tasks.

Security software may encrypt packets. Encryption may include the source IP address and port, preventing other devices and servers from reading the source address of packets.

Extensible Service Provider Accepts Network-Service Plugins—FIG. 4

The parent application, U.S. Ser. No. 09/042,306, hereby incorporated by reference, discloses in detail an extensible service provider that manages and orders network-service plugins. FIG. 4 is a diagram of a network architecture using an extensible service provider that manages and orders network-service plugins. The architecture is based on Winsock-2, the second-generation network architecture for Microsoft's Windows operating systems. Winsock-2 provides connections or "sockets" for high-level applications to connect to a network. A socket is the identifier for a given connection, or for a connectionless data-gram flow.

High-level applications 32 send and receive information to a network by making calls to Winsock-2 library 34. Winsock-1 calls from high-level applications are also routed through in a similar fashion. These calls use an applications-programming interface (API) that defines the function calls and their syntax. Winsock-2 library 34 is a dynamic-link library (DLL) of these function calls and other network-support routines.

Earlier versions of Winsock communicated directly with the lower TCP layer 40, which provides a Transmission Control Protocol for establishing sessions with remote hosts over a network. TCP layer 40 sends data to IP layer 42, which splits the data into Internet-Protocol IP packets and adds header information such as the source and destination IP address. IP layer 42 sends and receives these IP packets to the network media using physical layers such as a media-access controller (MAC).

While direct communication from Winsock-2 library 34 to TCP layer 40 can occur, Winsock-2 provides a service-provider interface (SPI) to third-party software modules known as layered providers or layered service providers. Instead of having many layered providers all communicating with TCP layer 40, a single extensible service provider 50 is installed. Extensible service provider 50 intercepts all network traffic at a lower level than the applications. Extensible service provider 50 fits between Winsock-2 library 34 and TCP layer 40, operating on data sent from Winsock-2 library 34 to TCP layer 40 for transmission. Extensible service provider 50 is "extensible" since it allows for the expansion of network services.

Extensible service provider 50 manages or controls the execution of additional network services provided by plugins 52. Plugins 52 are reduced in size and complexity compared with layered service providers because overhead functions and filtering is performed for all plugins by extensible service provider 50.

Plugins 52 provide various extra network services such as encryption, compression, security, proxies, or re-routing. These network services are transparent to high level applications 32 and can be activated for all applications using the network. Extensible service provider supplies a framework for managing and ordering a wide variety of plugin services.

Policy servers are desirable for prioritizing and regulating network traffic. Policy servers can better prioritize IP packets when the originating application and user are known. Although IP packets do not identify the high-level application or user that sent the data, this information is useful to policy servers. It is desired to collect information such as the originating application and user from client and server machines that can be used by the policy server in making priority decisions.

However, information must be collected from the client and server machines in a manner that is transparent to high-level applications. It is desired to install plugins on client and server machines that can be queried by the policy server. A specialized plugin for the extensible service provider is desired that monitors and collects information on network traffic from a client or server. This information includes the originating and destination high-level applications, data rates, and users. It is desired to prioritize network traffic based on high-level applications and users rather than low-level IP addresses and TCP ports.

SUMMARY OF THE INVENTION

A client-side application-classifier has an upper interface to a higher-level network-socket library. The higher-level network-socket library provides high-level network functions to high-level user applications by generating a socket for connecting to a remote machine on a network. A lower interface is to a network-transport layer that formats data for transmission over the network. An interceptor is coupled between the upper and lower interfaces. It intercepts network events.

An examiner is coupled to the interceptor. It examines the network event intercepted and collects statistical information about the network event. The statistical information includes:

an application name of one of the high-level user applications that caused the network event;

a timestamp for the network event;

- a byte count when the network event is a transfer of data over the network;
- Internet addresses and ports when the network event is a connection or a data transfer; and
- a process identifier of a running instance of the high-level user application.

A consolidator is coupled to the examiner. It consolidates the statistical information into application-classifier tables. The application-classifier tables include current tables for currently-running instances of applications, and historical tables that include closed applications. A reporter is coupled to the consolidator. It sends the statistical information from the application-classifier tables to a remote policy server on the network. The statistical information includes the application name. Thus the statistical information for network events is collected by the client-side application-classifier.

In further aspects of the invention the interceptor is an extensible service provider and the examiner is an application-classifier plugin to the extensible service provider. The extensible service provider controls other plugins providing low-level network services.

In still further aspects the examiner generates an event object containing the statistical information. The event object is sent to the consolidator and written into the application-classifier tables. The network event is:

- an application startup event when a high-level application is initialized;
- an application cleanup event when the high-level application is terminated;
- a socket open event when a new socket is opened;
- a socket close event when a socket is closed;
- a connect event when a connection is made from a client to a remote server;
- an accept event when a connection is accepted from a remote client;
- a send-complete event when a flow of data has been sent from the client to the remote server; and
- a receive-complete event when a flow of data has been sent from the remote server to the client.

In further aspects the statistical information for all network events includes a process identifier. The application-classifier tables are indexed by the process identifier. The application-classifier tables store for each flow of each high-level application:

- the process identifier;
- the timestamp;
- the application name;
- the byte count when the network event is a transfer of data over the network; and
- Internet addresses and ports when the network event is a connection or a data transfer;
- and wherein an application-classifier table for a high-level application contains:
  - maximum, average, and most-recent data-transfer rates for flows generated by the high-level application.

In other aspects the network-transport layer is a TCP/IP data communications stack coupled to a first network through a first media-access control (MAC) adapter and coupled to a second network through a second MAC adapter. The client-side application-classifier also has a network enhancer that is coupled between the network-transport layer and the first and second media-access controllers. It intercepts network packets and extracts routing information including source and destination network addresses.

A route table is coupled to the network enhancer to store the routing information for the network packets. The examiner is coupled to the route table to determine a source address of either the first MAC adapter or of the second MAC adapter or other MAC adapters when the source address is not available from the upper interface. Thus source addresses for clients with two network connections is obtained by the network enhancer below the TCP/IP stack.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 illustrates differing priorities of various kinds of network traffic.

FIG. 2 shows an Internet packet.

FIG. 3 is a diagram of a network that controls traffic using policy rules.

FIG. 4 is a diagram of a network architecture using an extensible service provider that manages and orders network-service plugins.

FIG. 5 is a diagram of an application-classifier that plugs into an extensible service provider for network services.

FIG. 6 is a table of events that trigger the application-classifier plugin to send data objects to the controller for classification.

FIGS. 7A–7E show definitions for objects that transfer data about network events from the application-classifier plugin to the controller for classification and storage.

FIG. 8 shows the current and historical tables of network events maintained by the application-classifier consolidator.

FIGS. 9A, 9B are diagrams of the format of an entry in the consolidator's tables.

FIG. 10 illustrates dual-level application-classifier plugins when multiple network addresses are used by a client.

FIG. 11 shows how the application-classifier plugin is useful for providing information for policy control applications in a policy server.

FIG. 12 is a diagram of a network using policy rules determined by queries of application-classifier plugins installed on clients.

DETAILED DESCRIPTION

The present invention relates to an improvement in network policy servers and their clients. The following description is presented to enable one of ordinary skill in the art to make and use the invention as provided in the context of a particular application and its requirements. Various modifications to the preferred embodiment will be apparent to those with skill in the art, and the general principles defined herein may be applied to other embodiments. Therefore, the present invention is not intended to be limited to the particular embodiments shown and described, but is to be accorded the widest scope consistent with the principles and novel features herein disclosed.

High-Level Better Than Low-Level for Prioritization

The inventors have realized that policy servers typically prioritize network traffic based on low-level IP addresses and TCP ports. Low-level prioritization is undesirable because IP addresses and TCP ports are often shared by many applications and users. All applications and users sharing IP addresses and ports would be given the same priority. Even when IP addresses are statically assigned to one machine, all applications on that machine may need to be given the same priority, even though some applications are inherently more important than others.

Ideally, a user is given high priority when using corporate applications such as client databases and sales forms, but lower priority when using personal, non-business applications such as web browsing or graphics downloading. This requires that the names of the high-level applications and users be used for prioritizing network traffic, not just the IP addresses and TCP ports.

The names of the high-level application and its users on the client and on the server machines can be collected using the invention. The high-level application names are then used by a policy server to prioritize IP packets from these applications, rather than only use IP addresses and TCP ports. The high-level application and user names are then associated with a traffic signature. The traffic signature is a unique representation of the user's application network traffic which can include the IP address, TCP port, and some of the data in the packets. Packets matching the traffic signature can then be identified at various points in a network and acted upon. Thus network traffic is prioritized by user and application, rather than by machine.

Since a policy server cannot consistently identify a high-level application by examining the IP packets passed through the network, the invention identifies network traffic by application and user. A software module on client or server machines collects information on network traffic from each client or server. A plugin for the extensible service provider on a client or server machine collect statistics on network traffic that can be read by a policy server. The collected statistics are arranged by user and high-level application and include data rates and time stamps.

Current Invention is a Plugin Module for ESP of Parent Application

The parent application, U.S. Ser. No. 09/042,306, hereby incorporated by reference, discloses in detail an extensible service provider (ESP) that manages and orders network-service plugins. The current invention includes a plugin module for the extensible service provider described in the parent application. This plugin module is used to collect network-traffic statistics and identify traffic signatures for applications. Since the traffic signatures collected include the name of the high-level application, the plugin is known as an application-classifier engine (ACE) plugin.

The application-classifier plugin resides on client and/or server machines and each collects network statistics for packets originating from or received by the machine. A policy server gathers information from the machines by polling the application-classifier plugin in each client and reading the collected statistics. Alternately, each client can periodically send the collected information to the policy server. The policy server can then make policy decisions and prioritize network traffic based on the information collected from the clients' application-classifier plugins.

Application-Classifier Plugs Into Extensible Service Provider—FIG. 5

FIG. 5 is a diagram of an application-classifier that plugs into an extensible service provider for network services. High-level applications 32 send and receive information to a network by making calls to Winsock-2 library 34. These calls use an applications-programming interface (API) that defines the function calls and their syntax. Winsock-2 library 34 is a dynamic-link library (DLL) of these function calls and other network-support routines.

TCP layer 40 sends data to IP layer 42, which splits the data into Internet-Protocol IP packets and adds header information such as the source and destination IP address. IP layer 42 sends and receives these IP packets to the network media using physical layers such as a MAC adapter. Multiple high-level applications 32 can spawn multiple processes that use multiple stacks such as TCP/IP and other protocols (not shown).

Instead of having many layered providers communicating with each other and TCP layer 40, a single extensible service provider 50 is installed, as described in detail in the parent application. Extensible service provider 50 intercepts all network traffic at a lower level than the applications. Extensible service provider 50 fits between Winsock-2 library 34 and TCP layer 40, operating on data sent from Winsock-2 library 34 to TCP layer 40 for transmission. Extensible service provider 50 is "extensible" since it allows for the expansion of network services.

Extensible service provider 50 manages or controls the execution of additional network services provided by plugins. Plugins are reduced in size and complexity compared with layered service providers because overhead functions and filtering is performed for all plugins by extensible service provider 50.

Plugins provide various extra network services such as encryption, compression, security, proxies, or re-routing. These network services are transparent to high level applications 32 and can be activated for all applications using the network. Extensible service provider supplies a framework for managing and ordering a wide variety of plugin services.

One such plugin is application-classifier plugin 51. Other plugins (not shown) may also be installed and controlled by extensible service provider 50 and thus co-exist with application-classifier plugin 51. One or more filters can be performed by extensible service provider 50 to reduce the amount of traffic that activates application-classifier plugin 51. In the preferred embodiment no filters are used so that all network traffic activates application-classifier plugin 51. Thus network statistics are collected on all traffic passing through extensible service provider 50 to and from TCP layer 40.

Application-classifier plugin 51 is a Windows DLL that is loaded by extensible service provider 50 on initialization. Extensible service provider 50 reads a list of plugins to load from the Windows registry and loads all the listed plugins, including application-classifier plugin 51. When application-classifier plugin 51 is loaded, it attaches itself to one or more of the filters registered with extensible service provider 50. In the preferred embodiment, application-classifier plugin 51 attaches itself to a universal filter that activates application-classifier plugin 51 for all network traffic.

When an event occurs, such as a command or call from Winsock-2 library 34 or from TCP layer 40, or completion of network I/O, extensible service provider 50 compares the properties of the event to each of its filters and generates an event activating all plugins attached to matching filters.

Application-Classifier Components

When an event activates application-classifier plugin 51, a data object containing details of the event is sent to controller 62. Controller 62 classifies the event and updates tables maintained by application-classifier-engine (ACE) consolidator 60. Consolidator 60 keeps tables of the most-recent events for each application and process, but also keeps historical tables of past events, even for applications that have closed their connections and sockets. Statistical fields in the tables are also updated, such as the number of bytes sent by an application or any of its processes, and timestamps when connections were opened and closed.

A remote policy server can access the tables stored by consolidator **60** by connecting with DCOM server **64**. DCOM server **64** contains proxy and stub objects that hide the details of network communication from higher-level application programs. DCOM server **64** uses Microsoft's Distributed-Component-Object Model (DCOM) of distributed programming objects. Winsock-2 library **34** receives calls from DCOM server **64** that send and receive data packets over the network, passing down the protocol stack through TCP layer **40** and IP layer **41**. When a remote policy server connects as a client to DCOM server **64**, a request can be sent from the remote policy server, over the network to controller **62**. The request can instruct controller **62** to fetch one or more of the tables stored by consolidator **60** and send them back over the network to the policy server. Controller **62** and DCOM server **64** can be combined into a single controller module.

Real-time information may also be collected by a remote policy server. The policy server can register with controller **62** for certain kinds of events. For example, a policy server may need to know when a certain high-priority, high-bandwidth application such as videoconferencing opens a connection. The policy server can send the registration request through DCOM server **64** to controller **62**. Whenever a connection for the high-priority videoconferencing application is opened and the event received by application-classifier plugin **51**, controller **62**, on receiving the event object from application-classifier plugin **51**, sends the event object to DCOM server **64** to be immediately sent over the network to the policy server. The policy server can then reserve bandwidth for the videoconferencing application's packets before they arrive.

Objects Generated by Application-Classifier Plugin for Events—FIG. 6

FIG. **6** is a table of events that trigger the application-classifier plugin to send data objects to the ACE controller for classification. Events are generated by the Winsock-2 library when an application starts or ends, when it opens or closes a socket, when the connection state of the socket changed, and when data is sent or received.

Events can be grouped into three categories. Application events occur when a high-level networking application on the client machine opens or closes. Socket events are generated when a socket is opened, closed, changes its connection state, or the data is sent or received. A flow event occurs when a datastream begins or ends.

FIG. **6** shows that the EventAppStart object is generated by the application-classifier plugin when an application startup event occurs. A startup event is signaled by the extensible service provider by activating the application-classifier plugin at the StartUp entry point. The EventAppStop object is generated when an application cleanup event occurs, such as when the extensible service provider activates the application-classifier plugin at the Cleanup entry point. The application starting up or cleaning up makes a Winsock-2 API call, which is passed down to the extensible service provider which generates the proper entry point into the application-classifier plugin.

Either the AppStart or the AppStop object generated by the application-classifier plugin is sent to the ACE controller. The AppStart and AppStop objects use the EventAppInit data-object definition, shown in FIG. 7A.

Several different kinds of socket events can occur. The SocketOpen, SocketClose, SocketBind, SocketAccept, and SocketConnect objects are generated by the application-classifier plugin when one of the closely-named Winsock API functions is called. For example, when an application opens a socket with a WSASocket API call, Winsock-2 generates a WSPSocket SPI call to the extensible service provider, which activates the application-classifier plugin at the SocketOpen entry point. When the lower-level TCP layer determines that the connection with the remote machine has been made, the ConnectComplete event is generated.

Different kinds of objects are generated for socket events. The EventSocketInit data-object definition (FIG. 7B) is used to generate the SocketOpen and SocketClose objects sent to the ACE controller, while the EventSocketState data-object definition (FIG. 7C) is used to generate the SocketBind, SocketAccept, and SocketConnect objects when the socket state is changed and the BindComplete, AcceptComplete, and ConnectComplete events occur.

Completed data transfers signaled by the RecvComplete and SendComplete events generate the SocketIn and SocketOut objects, which are based on the EventSocketDataXfer data-object definition (FIG. 7D).

Flow events occur when a datastream completes. A flow is a connected TCP data stream or a sequence of unconnected (UDP) data-grams to and/or from a unique IP address and port. For TCP (connection oriented) steams, ConnectComplete signals the start of a flow in most cases. Some Winsock applications do not wait for completion of a connection: they simply initiate the connection and attempt to send or receive, assuming that data is transferred when the connection is made. In this special case, the triggering event for the flow is the combination of a ConnectComplete event and successful data transfer, signaled by RecvComplete and SendComplete events.

For unconnected UDP data-grams, the first data successfully sent or received from a unique IP address and TCP/UDP port defines the start of a flow. This is indicated by the RecvComplete and SendComplete events that generate the Socketin and SocketOut objects.

When the flow begins, either the ConnectComplete event for TCP or the RecvComplete and SendComplete events for UDP, the FlowStart object is generated, based on the EventFlowInit data-object definition (FIG. 7E). The end of the flow is signaled by the SocketClose event, which generates a FlowStop object also based on the EventFlowInit data-object definition **89**.

Data Objects—FIGS. 7A–7E

FIGS. 7A–7E show definitions for objects that transfer data about network events from the application-classifier plugin to the ACE controller for classification and storage. The triggering events for these objects were shown in FIG. 6. Each of the tables in FIGS. 7A–7E and 9A, 9B describe array data objects that are transferred.

FIG. 7A shows the EventAppInit data-object definition. A version number is the first parameter in the definition. A timestamp is stored for the time that the event activated the application-classifier plugin. The unique process ID for the Winsock process that originated the event is stored in the ProcessID field. One application can spawn several network processes, but each process is identified by a unique ID assigned by the Windows operating system.

The name of the high-level application is retrieved using the Win32 API library call GetModuleFileName. Thus the objects based on EventAppInit data-object definition include the names of the application, user, and host, as well as a timestamp and the unique process ID.

FIG. 7B shows the EventSocketInit data-object definition. A version number, timestamp, and the unique process ID are stored. The socket handle and the network-layer (IP, IPX) and transport-layer (TCP or UDP) protocols are also stored.

The application and user names are not contained in the socket objects since the unique process ID can be used to associate the information with the corresponding application and user. An event created by the process such as a StartUp event contained the application and user names. Thus the consolidator already has the association of the unique process ID to the application before any socket events occur.

FIG. 7C shows the EventSocketState data-object definition. A version number, object timestamp, and the unique process ID are stored. The socket parameter is the value of the socket handle assigned by the Winsock library. The local and remote addresses are also stored. These are the destination and source IP addresses and TCP/UDP ports. For SocketBind objects, the remote address is not yet known and the remote address field is set to zero.

The EventSocketDataXfer data-object definition is shown in FIG. 7D. Again, a version number, timestamp, and the unique process ID are stored. The socket address and the local and remote addresses are also stored. The number of bytes in the transfer is also stored in the Bytes parameter. The number of bytes transferred is a useful statistic, since the policy server can use it to determine the bandwidth used by the application. Bandwidth hogs can thus be identified.

FIG. 7E highlights the EventFlowInit data-object definition. Flows are sequences of data packets sent or received between two endpoints. The version number, timestamp, and the unique process ID are stored. A unique flow ID that was assigned by the ACE plugin is also stored, since each process can generate several flows.

The transport-layer and network-layer protocols are stored in the flow object. The source and destination IP addresses and TCP/UDP ports are also stored in the object.

Current and Historical Tables in Consolidator—FIG. 8

FIG. 8 shows the current and historical tables of network events maintained by the application-classifier consolidator. Statistical data stored in the tables include overall byte counts of data sent or received, average and maximum bytes sent/received per second, and start and stop timestamps of processes and flows.

Current tables 92, 94, 96 are snapshots of currently-running applications, processes, and flows. When a flow, process, or application closes, its table entry is deleted from the current tables. Thus a policy server reading current tables 92, 94, 96 might not see statistics for applications or flows that recently closed.

Historical tables 91, 93, 95 contain entries for both running and closed applications, processes, and flows. Historical tables 91, 93, 95 can contain only a limited number of entries; once the limit is reached, the oldest entry of all the applications, processes, or flows is deleted to make room for a new entry. Flow historical table 95 is more likely to fill up before process historical table 93 or application historical table 91, since a single application can generate several processes, and each process can generate many flows. Of course, the sizes of the tables can be adjusted to allow more room for flow table 95 and process table 93 than for application table 91.

Application tables 91, 92 each contain no more than one entry for each application. Network statistics are consolidated into the single entry for all processes and flows for the application. Thus a policy server can read an entry for a specific application to find out how many bytes the application has sent or received, regardless of how many connections have been made for the application. This allows high-bandwidth applications to be quickly identified. If two or more instances of an application are running, their network statistics are combined into a single entry in application table 92. Thus if a user as multiple browser windows running, the total browser traffic can be found in application table 92. Likewise, when an application is closed and restarted, a running total statistic for all previous instances of the application is kept. Restarting the application thus does not reset the usage statistics.

A finer granularity to network statistics is stored by process ID in process tables 93, 94. Each unique process ID contains one entry per table. When another instance of an application is started, a second process ID is assigned to it, and a second entry is created in process table 93. When an application is closed and later restarted, the restarted application has a different process ID than the application did before it was closed. Separate entries are stored in historical process table 93, although only the running processes have an entry in current process table 94.

Flow tables 95, 96 provide the finest granularity of network statistics. Each unique flow has its own entry. Different flow ID's are assigned to each flow generated by an application. Their entries are placed into historical flow table 95.

Storage space can be reduced when the consolidator stores only flow tables 95, 96. When the policy server or ACE controller reads an entry in the application or process tables, the entry can be generated on the fly by the consolidator. All flow-table entries for the requested process or application are read, and their entries merged. Byte counts are summed, and average and maximum transfer rates and start and stop times are calculated for the process or application. Since table queries are less frequent than updates, overall processing is reduced by storing just the finest-granularity tables and generating the coarser-table entries.

Consolidator Table Entries—FIGS. 9A, B

FIGS. 9A, 9B are tables of the format of an entry in the consolidator's tables. Again, a version number, the unique process ID and the flow ID are stored. For application entries, the most-recent process ID and most-recent flow ID are stored. Process entries store the most-recent flow ID. The application and user names stored at the end of the entry allow the application and user names to be found. The start and stop times of applications, processes, or flows are stored in historical tables. For current tables, the application, process, or flow has not yet terminated, so the stop time is set to zero. For historical tables, the stop time is the time the last process instance of the application or flow was terminated.

The transport protocol (TCP, UDP, etc.) is stored along with the source and destination IP addresses and TCP ports. For application and process tables, these are the addresses and protocols for the most-recent flow.

Statistical information, such as byte counts, is also stored. Separate fields store the total number of bytes sent and the total bytes received for the application, process, or flow. The maximum and the minimum number of bytes sent or received in any one-second period is stored in the MinRate and MaxRate fields. The average byte rate is stored as AvgRate.

A sample of the rate can be taken over a specified period. The number of bytes sent and received during the sample period is stored in DeltaBytesSent and DeltaBytesRecvd. The length of the sample period is stored as DeltaRate. In FIG. 9B, a flow table is shown. It contains an array of flows, with each flow as shown in FIG. 9A. All the flows on a machine can be counted using the flow table of FIG. 9B.

MAC-Level Route-Table Plugin—FIG. 10

FIG. 10 illustrates dual-level application-classifier plugins when multiple network addresses are used by a client. Some client machines connect to more than one network. For example, the client machine may have two Ethernet LAN cards, or a Token-Ring and an Ethernet card, or a Ethernet card and a dial-up modem connection. Two IP addresses are assigned to the client machine. Packets from TCP/IP layers **40**, **42** may be assigned either IP source address, depending on which network the data is sent out over. For unconnected UDP datagrams, Winsock is not notified by TCP/IP layers **40**, **42** of which IP address is used. Thus application-classifier plugin **51** is not able to determine the source IP address for this special case.

Network enhancer **68** is a low-level network driver installed below IP layer **42** but above media-access-controller MAC drivers **70**, **70'**. MAC drivers **70**, **70'** are software drivers that transfer data to network adapter cards on the client machine and control the cards. Two different network connections are made by MAC drivers **70**, **70'**, each using a different source IP address.

Network enhancer **68** is installed by the operating system as a low-level driver (a Network Device Interface Specification NDIS shim) that intercepts all outgoing and incoming network traffic. A filter is used by network enhancer **68** to intercept only ARP Address Resolution Protocol data that is sent prior to a datagram, since the ARP protocol specifies the destination and source IP addresses. The filter activates route-table plugin **66** when ARP traffic is detected. Route-table plugin **66** is a plugin module that receives the IP addresses from network enhancer **68**. The destination and source IP addresses are captured by route-table plugin **66** and stored in a route table of source and destination IP addresses by route-table plugin **66**.

When an unconnected datagram is sent through extensible service provider **50** to TCP/IP layers **40**, **42**, application-classifier plugin **51** queries route-table plugin **66** for the source IP address used by TCP/IP layers **40**, **42**.

Application-classifier plugin **51** then generates the event object using the source IP address obtained from route-table plugin **66**, and sends the event object to controller **62**. Controller **62** classifies the event object and updates the application, process, or flow tables in ACE consolidator **60**. A remote policy server can read the table objects of consolidator **60** through DCOM server **64** as described earlier.

Application-classifier plugin **51** queries route-table plugin **66** by performing a user-mode to kernel call. Such kernel calls can hurt performance. However, such a call is required only once for each new flow, and only for clients connected to two or more networks. An alternative is to store the route table in the Windows registry.

The inventors have recognized this unusual problem caused by Winsock-2 not providing the source IP address for unconnected datagrams. The problem is only apparent when two or more network adapters are connected to a system. Dropping prices of network adapter cards and use of modems may make dual-network systems more common in the future.

Plugins Useful for Policy Control—FIG. 11

FIG. **11** shows how the application-classifier plugin is useful for providing information for policy control applications in a policy server. The application-classifier plugin can periodically send the collected statistics to a server by creating network calls to send packets containing the statistics. The application-classifier plugin is transparent to higher-level applications since it operates between the Winsock-DLL called by the applications and the lower-level TCP/IP stack.

Policy control is implemented by policy server **18**, which collects detailed statistics of network traffic from various network nodes such as client **10**. High-level applications **32** in client **10** include www browsers, corporate database viewers and data-entry terminal apps, and workgroup apps such as network-enabled word processors and spreadsheets and other editors. Winsock-2 library receives calls from applications **32** for access to the network. These calls are passed down to extensible service provider **50**, which filters these calls and executes one or more plugins.

Application-classifier plugin **51** intercepts and analyzes the network traffic from client **10**. The analyzed or modified data from application-classifier plugin **51** is sent back to extensible service provider **50** and sent down through TCP layer **40** and IP layer **42** and finally out over the network.

Each event, such as opening or closing a socket, making or breaking a connection, or transferring data packets is intercepted by application-classifier plugin **51**. The event is analyzed and classified by application-classifier controller **62**, which updates tables in application-classifier consolidator **60**. Thus statistics on network traffic from client **10** are stored in the tables of application-classifier consolidator **60**.

The Winsock-2 API is provided for use by the plugins. This API is useful for allowing the plugins to communicate with remote servers such as the policy server. For example, application-classifier controller **62** can use Winsock calls to send ("push") data over the network using TCP layer **40"** and IP layer **42"**.

Policy server **18** may observe some packets from client **10**, but does not know what priority to assign to them. A policy application **32** can make a call to Winsock-2 library **34'** that is sent through extensible service provider **50'**, TCP layer **40'**, and IP layer **42'** to client **10** over the network. This call can be directed to DCOM server **64** in client **10**, asking what kind of traffic was recently sent. DCOM server **64** reads the tables in application-classifier consolidator **60** to obtain the desired information. DCOM server **64** then responds to policy server **18** by sending the desired information, such as a log of packets sent, together with their applications **32** that generated the packets and other high-level information that may not be contained in the packets sent.

Other information may be requested by policy server **18**, such as the number of bytes sent by any particular application **32** on client **10**, or the total number of packets sent in a time period. The highest burst rate of packets or the average packet transmission rate can also be obtained.

DCOM server **64** allows easier object-oriented programming techniques to be used, hiding the details of network communication from higher-level programmers. Stub and proxy objects are used on local and remote machines to facilitate communication over the network. Extensible service provider **50'** on policy server **18** is not required and can be eliminated in some embodiments.

Thus policy server **18** is able to query client **10** by using application-classifier plugin **51**, which analyzes and logs network traffic from client 10. More sophisticated schemes could have policy server 18 deciding that the traffic from client 10 is low-priority, and instructing application-classifier plugin 51 or another traffic-blocking plugin (not shown) to block traffic from a particular application 32 in client 10. Thus network traffic can be blocked from the source.

The policy enforcer (policy server) is able to identify the user and the high-level application of network packets by polling the application-classifier on the client machine. The IP address and TCP port are sent from the policy server to lookup the packet's user and application in the history tables kept by the application-classifier consolidator.

Plugins enable policy controls to be implemented in a manner transparent to higher-level applications. Plugins provide a way for network software to more closely examine network traffic from any machine, and even exert control over that machine's traffic.

Policy Server Queries Clients—FIG. 12

FIG. 12 is a diagram of a network using policy rules determined by queries of application-classifier plugins installed on clients. Policy server 18 queries application-classifier plugins installed on client PC's 10, 12, as described for FIG. 5. The DCOM servers on client PC's 10, 12 are used by policy server 18 to connect and read the ACE tables stored by the consolidators. Thus policy server 18 is able to determine which high-level application and which user is sending packets through edge device 14 by performing a lookup of the ACE tables on client PC's 10, 12 for the flow table having the source and destination IP addresses and TCP ports of the packets seen at edge device 14. The application and user names are stored with the matching entry in the flow table.

Policy server 18 can also find bandwidth-hogging applications by reading all historical application ACE tables on client PC's 10, 12, and comparing the average and maximum byte rate fields to acceptable thresholds. Applications with high transmission rates can be identified, and policy server 18 can instruct edge device 14 to block or delay packets for flows originating from these applications. The flows for an application can be read from the ACE tables.

Client PCs 10, 12 send IP packets over local network 15 to corporate server 16 and Internet 20. Edge device 14 is a router, switch, gateway, modem or other network device that connects local network 15 to Internet 20. Edge device 14 is able to block or delay packets to and from Internet 20 so that higher-priority packets experience less delay than lower-priority packets. Edge device 14 may examine packets and apply policy rules to determine which packets to accelerate and which to delay.

Policy server 18 can still send a policy query to corporate server 16, which may or may not have the application-classifier plugin installed. Corporate server 16 can respond that certain of its packets are high or low priority. Also, corporate server 16 may indicate if an IP address is high or low priority.

Low-priority web browsing from client 10 can be identified by finding the application name in the flow tables for the IP address for client 10 and port 80 used by the browser. Even dynamically assigns ports or IP addresses can be associated with their sending applications.

ADVANTAGES OF THE INVENTION

The names of the high-level application and its associated user on the client and on the server machines can be collected using the invention. The high-level application names then can be used for prioritizing IP packets from these applications, by associating the application name and its associated flows with traffic signatures. Thus network traffic is prioritized by user and application, rather than by machine. A user may have high priority when using corporate applications such as client databases and sales forms, but lower priority when using personal, non-business applications such as web browsing or graphics downloading.

Information can be collected from the client machines in a manner that is transparent to high-level client applications. Plugins installed on client machines can be queried by the policy server. A specialized application-classifier ACE plugin for the extensible service provider monitors and collects information on network traffic from a client. This information includes the originating and destination high-level applications, data rates, and users. Network traffic is prioritized based on high-level applications rather than low-level IP addresses and TCP ports.

The layered network architecture of the extensible service provider allows multiple third-party service providers to be installed in addition to the application-classifier ACE plugin. The Winsock-2 architecture is expanded for network services provided at a low level. These network services can transparently intercept network traffic. The complexity of layered providers is reduced and redundant filtering by each layered provider is eliminated by using the extensible service provider. An expandable system that manages, organizes, and orders low-level network service providers is attained. Plugins are executed in a functionally correct order even when many layered service provider plugins from different vendors are installed.

These plugins are simplified compared with Winsock-2 layered service providers, since overhead for communication with the Winsock-2 library and the TCP layer are handled by the extensible service provider. Since there are many Winsock-2 functions that are not used by most plugins, the overhead for these seldom-used functions is contained in the extensible service provider, reducing the complexity of the plugins. Complex I/O such as blocking, non-blocking via messages or events, and overlapped are handled by the ESP.

ALTERNATE EMBODIMENTS

Several other embodiments are contemplated by the inventors. For example, many software implementations using many different programming languages are possible. The invention may be adapted for UNIX and other operating systems, as well as future versions of Windows. Indeed, UNIX-DCOM servers are now appearing, allowing a UNIX policy server to access the application-classifier tables residing on a Windows machine. Rather than use distributed objects, the information can be sent using a management protocol such as SNMP or via FTP.

Edge devices may also be installed at internal points in a network, such as between sub-nets in a corporate Intranet. Many edge devices can be employed in a single network. Indeed, the invention can be applied to traffic within a local network or within an Intranet. One or more policy servers collect information from application-classifiers on clients or at intermediate points, and control these edge devices. Routers, bandwidth control boxes, switches, firewalls, and Virtual Private Networking (VPN) boxes, as well as Network Management systems can use need the information that the application-classifier can provide. These intermediate devices can be integrated with clients or servers using the invention.

Edge devices may not only limit traffic from certain applications identified by the application-classifier, but modify the traffic as well. For example, some applications may transmit confidential or sensitive information. The edge devices can encrypt packets for flows from such applications identified by the application-classifier. The application-classifier plugin itself, or another installed plugin can be used to block network traffic from the client, rather than the edge device. The policy server can program the plugin on the client machine to block packets from certain low-priority applications, or block after a threshold number of bytes have been sent in a time period. Thus network traffic can be blocked at the source, the client PC.

The invention can also be used for other classification purposes. Identifying application traffic enables security, access, routing, discard, and other tasks. Decisions can be made either remotely through a policy server or infrastructure device (gateway, router, bandwidth control device, or switch) or locally on the host system itself. In a very simple case, the invention can be used locally to effectively include the policy server on the same system. This approach might enable a user of a cable modem or DSL service to request a higher level of service from a network (and promise to pay accordingly). Usually, this selection of service would be done per application, e.g. if the user wanted to receive real time video, invoke IP telephony, or just get a file transferred more quickly.

Security applications can examine outgoing packets and encrypt packets only from certain applications, such as financial applications. Other applications from the same IP address, such as e-mail, can be skipped and sent out without encryption.

Other network-based, distributed-object programming standards besides DCOM can be substituted, such as CORBA or COPS. Future improvements to Winsock-2 can also be used with the invention. Earlier versions of Winsock communicated with both TCP and UDP, and UDP can be substituted for the TCP layer with the invention.

The invention has been described as filtering packets. However, data may not yet be divided into the final packets transmitted over the network media when intercepted by the extensible service provider. The data sent from the Winsock-2 library functions down to the extensible service provider and the TCP layer may be further divided by the TCP and IP layers into smaller packets for transmission. Thus the term "packets" when used for the extensible service provider do not strictly refer to the final transmitted packets, but to the data and header information that will eventually form one or more packets.

The invention can also work with other protocols such as SNA, IPX, X.25, etc. It is not restricted to IP (TCP, UDP, etc). The invention is extensible to perform more granular classification of traffic. For example, print and file transfer traffic may be mixed with interactive traffic sent from a single application. The invention can break these out into unique flows, identifying the traffic signatures for each 'subflow'. Two examples of this are tn3270 which combines terminal traffic, print traffic and file transfer traffic in a single telnet session, and a SAP/R3 application which has a variety of financial transaction types (including printing). The invention can provide extensions to identify the traffic signatures of individual transactions.

The foregoing description of the embodiments of the invention has been presented for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise form disclosed. Many modifications and variations are possible in light of the above teaching. It is intended that the scope of the invention be limited not by this detailed description, but rather by the claims appended hereto.

We claim:

1. A client-side application-classifier comprising:

an upper interface to a higher-level network-socket library, the higher-level network-socket library for providing high-level network functions to high-level user applications by generating a socket for connecting to a remote machine on a network;

a lower interface to a network-transport layer, the network-transport layer for formatting data for transmission over the network;

an interceptor, coupled between the upper and lower interfaces, for intercepting network events;

an examiner, coupled to the interceptor, for examining the network event intercepted and collecting statistical information about the network event, the statistical information including:

an application name of one of the high-level user applications that caused the network event;

a timestamp for the network event;

a byte count when the network event is a transfer of data over the network;

Internet addresses and ports when the network event is a connection or a data transfer; and

a process identifier of a running instance of the high-level user application;

a consolidator, coupled to the examiner, for consolidating the statistical information into application-classifier tables, the application-classifier tables including current tables for currently-running instances of applications, and historical tables that include closed applications; and

a reporter, coupled to the consolidator, for sending the statistical information from the application-classifier tables to a remote policy server on the network, the statistical information including the application name,

whereby the statistical information for network events is collected by the client-side application-classifier.

2. The client-side application-classifier of claim 1 wherein the interceptor is an extensible service provider and wherein the examiner is an application-classifier plugin to the extensible service provider, the extensible service provider for controlling other plugins providing low-level network services.

3. The client-side application-classifier of claim 1 wherein the examiner includes means for generating an event object containing the statistical information, the event object sent to the consolidator and written into the application-classifier tables.

4. The client-side application-classifier of claim 1 wherein the network event is selected from the group consisting of:

an application startup event when a high-level application is initialized;

an application cleanup event when the high-level application is terminated;

a socket open event when a new socket is opened;

a socket close event when a socket is closed;

a connect event when a connection is made from a client to a remote server;

an accept event when a connection is accepted from a remote client;

a send-complete event when a flow of data has been sent from the client to the remote server; and a receive-complete event when a flow of data has been sent from the remote server to the client.

5. The client-side application-classifier of claim 4 wherein the statistical information for all network events includes a process identifier, wherein the application-classifier tables are indexed by the process identifier.

6. The client-side application-classifier of claim 5 wherein the application-classifier tables store for each flow of each high-level application:

the process identifier;

the timestamp;

the application name;

the byte count when the network event is a transfer of data over the network; and

Internet addresses and ports when the network event is a connection or a data transfer;

and wherein an application-classifier table for a high-level application contains:

maximum, average, and most-recent data-transfer rates for flows generated by the high-level application.

7. The client-side application-classifier of claim 1 wherein the network-transport layer is a TCP/IP stack coupled to a first network through a first media-access controller and coupled to a second network through a second media-access controller, the client-side application-classifier further comprising:

a network enhancer, coupled between the network-transport layer and the first and second media-access controllers, for intercepting network packets and extracting routing information including source and destination network addresses; and

a route table, coupled to the network enhancer, for storing the routing information for the network packets;

the examiner coupled to the route table to determine a source address of either the first media-access controller or of the second media-access controller when the source address is not available from the upper interface,

whereby source addresses for clients with two network connections is obtained by the network enhancer below the TCP/IP stack.

8. A computer-implemented method for classifying network flows from a client, the method comprising:

calling a socket function for opening or transmitting data through a socket-connection for connecting a high-level application to a remote machine on a network, the socket function being a function in an applications-programming interface (API) used by high-level applications to access the network;

activating an extensible service provider before the data is sent from the socket function to a lower network-transport layer, wherein the data is intercepted by the extensible service provider, the extensible service provider for evaluating filters to determine which plugins need to be executed;

activating an application-classifier plugin attached to the extensible service provider before the data is sent to the network-transport layer;

collecting statistical information including a name of the high-level application generating the data, a user name, a timestamp, and a number of bytes transmitted when the application-classifier plugin is activated;

consolidating the statistical information collected by the application-classifier plugin in application-classifier tables; and

sending the statistical information to a policy server on a remote machine on the network, wherein the policy server prioritizes the data using the name of the high-level application obtained from the application-classifier plugin on the client,

whereby the policy server prioritizes network data based on names of high-level applications obtained from the application-classifier plugin on the client.

9. The computer-implemented method of claim 8 wherein the step of sending the statistical information comprises:

searching the application-classifier tables for matching entries having a source and a destination IP address that match a source and a destination IP address that the policy server obtained by examining a network packet, the network packet not containing the name of the high-level application; and

reading the name of the application from the matching entries and sending the name of the high-level application to the policy server as the high-level application that generated the network packet examined by the policy server,

wherein the policy server prioritizes network traffic based on high-level applications rather than low-level IP addresses.

10. The computer-implemented method of claim 9 further comprising:

generating an event object when the application-classifier plugin is activated, the event object indicating a type of network activity performed by the socket function, the event object containing the statistical information;

sending the event object to the application-classifier tables, the statistical information being added to the application-classifier tables.

11. The computer-implemented method of claim 10 further comprising:

finding bandwidth-hogging applications by reading byte-count fields in the application-classifier tables and comparing the byte-count fields to a threshold,

wherein applications with network flows having byte-counts above the threshold are identified as high-bandwidth applications.

12. The computer-implemented method of claim 11 further comprising:

using the timestamp in the statistical information and the number of byte transmitted to determine a rate of byte transfer;

storing the rate of byte transfer in the application-classifier tables.

13. The computer-implemented method of claim 8 wherein the application-classifier plugin is transparent to high-level applications, the application-classifier plugin performing low-level network services.

14. A computer-program product comprising:

a computer-usable medium having computer-readable program code means embodied therein for classifying network traffic according to high-level application name, the computer-readable program code means in the computer-program product comprising:

socket means for receiving data for transmission over a network, the data from a high-level application that uses a high-level library of socket-functions for sending the data to the socket means;

transport means for sending the data to a lower-level network-transport layer, the lower-level network-transport layer for formatting the data for transmission over the network; and

extensible service provider means, coupled to the socket means and to the transport means, for activating a application-classifier plugin when the data is sent to the transport means, the extensible service provider means further for activating other plugins;

the application-classifier plugin including means for collecting information about the data, the information including a name of the high-level application generating the data, a source address and a destination address, and a timestamp;

whereby the data is classified by the name of the high-level application generating the data sent to the network.

15. The computer-program product of claim **14** wherein the computer-readable program code means further comprises:

a consolidator, coupled to the application-classifier plugin, for storing the information collected in application-classifier tables with information collected for network data transmissions for other high-level applications,

whereby the information is stored in the application-classifier tables.

16. The computer-program product of claim **15** wherein the computer-readable program code means further comprises:

reporting means, coupled to the consolidator, for receiving requests from a policy server on a remote machine on the network, for reading the application-classifier tables and returning to the policy server the name of the high-level application from the application-classifier tables,

whereby the policy server looks up the name of the high-level application sending the data to the network.

17. The computer-program product of claim **16** wherein the request from the policy server includes source and destination IP addresses from data packets sent over the network from the socket means, but the data packets do not contain the name of the high-level application sending the data,

whereby the policy server cannot obtain the name of the high-level application from the data packets but only from the application-classifier tables.

18. The computer-program product of claim **16** wherein the computer-readable program code means further comprises:

filtering means for comparing transmission information for the data from the socket means to predetermined transmission criteria, for indicating when a socket matches the predetermined transmission criteria;

wherein the extensible service provider means only activates the application-classifier plugin when the socket matches the predetermined transmission criteria.

19. The computer-program product of claim **18** wherein the computer-readable program code means further comprises:

a blocking plugin, coupled to the extensible service provider means, for blocking the data from being transmitted to the network;

wherein the policy server determines which data is low-priority data by reading the names of high-level applications from the application-classifier tables;

wherein the blocking plugin blocks low-priority data from being transmitted on the network to reduce network traffic, the blocking plugin under control of the policy server,

whereby the low-priority data is blocked at the source before being sent over the network.

20. The computer-program product of claim **16** wherein the application-classifier plugin and extensible service provider means are installed on a client machine,

whereby the client machine collects the information for use by the policy server.

* * * * *

US006510164B1

(12) **United States Patent**
Ramaswamy et al.

(10) **Patent No.: US 6,510,164 B1**
(45) **Date of Patent: *Jan. 21, 2003**

(54) **USER-LEVEL DEDICATED INTERFACE FOR IP APPLICATIONS IN A DATA PACKET SWITCHING AND LOAD BALANCING SYSTEM**

(75) Inventors: **Kumar Ramaswamy**, San Jose, CA (US); **Cher-Wen Lin**, Milpitas, CA (US); **Randall David Rettberg**, Danville, CA (US); **Mizanur Mohammed Rahman**, Cupertino, CA (US)

(73) Assignee: **Sun Microsystems, Inc.**, Palo Alto, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **09/193,651**

(22) Filed: **Nov. 16, 1998**

(51) **Int. Cl.**^{7} **H04J 3/16**

(52) **U.S. Cl.** **370/466**; 370/235; 370/353; 370/401; 370/463; 709/321; 709/223

(58) **Field of Search** 370/229, 235, 370/351, 352, 353, 354, 356, 389, 392, 400, 401, 428, 463, 465, 466, 449; 709/521, 322, 323–327, 223, 312, 321

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,535,199	A	*	7/1996	Amri	370/392
5,581,729	A		12/1996	Nishtala et al.	
5,634,068	A	*	5/1997	Nishtala et al.	711/141
5,644,753	A		7/1997	Ebrahim et al.	
5,655,100	A	*	8/1997	Ebrahim et al.	711/144
5,657,472	A		8/1997	Van Loo et al.	
5,768,510	A	*	6/1998	Gish	709/203
5,771,349	A		6/1998	Picazo, Jr. et al.	395/188.01
6,006,275	A	*	12/1999	Picazo, Jr. et al.	709/249
6,246,680	B1	*	6/2001	Muller et al.	370/389
6,252,878	B1	*	6/2001	Locklear, Jr. et al.	370/401
6,253,230	B1	*	6/2001	Couland et al.	709/203
6,272,136	B1	*	8/2001	Lin et al.	370/392
6,272,522	B1	*	8/2001	Lin et al.	709/200

OTHER PUBLICATIONS

"U-Net: A User-Level Network Interface For Parallel And Distributed Computing" by Eicken et al., ACM Press, Operating Systems Review, vol. 29, No. 5, Dec. 1995, pp. 40–53.

"IP/ATM: A Strategy For Integrating IP With ATM" By Guru Parulkar, Douglas C. Schmidt, and Jonathan S. Turner, Computer Communications Review, US, Association For Computing Machinery, New York, vol. 25, No. 4, Oct. 1, 1995, pp. 49–58.

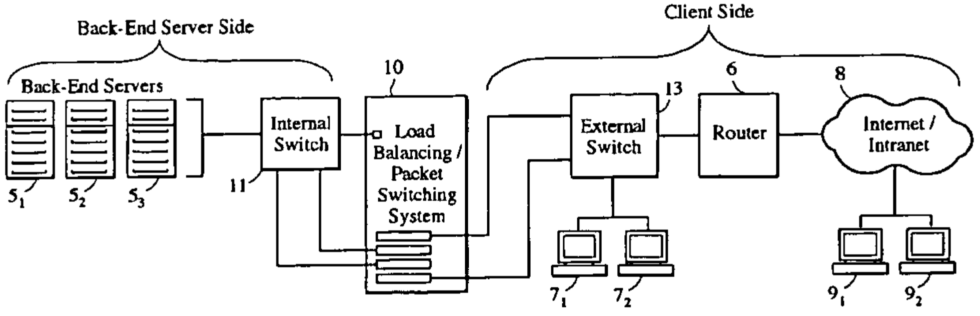
* cited by examiner

Primary Examiner—Ricky Ngo
Assistant Examiner—Tri H. Phan
(74) *Attorney, Agent, or Firm*—O'Melveny & Myers LLP

(57) **ABSTRACT**

A multiprocessor computer system comprises a plurality of network interfaces each adapted to be coupled to respective external networks for receiving and sending data packets to and from remote devices coupled to the external networks via a particular communication protocol. The multiprocessor computer system further comprises a plurality of symmetrical processors including a control processor and at least one switching processor. The switching processor further includes at least one network application executing thereon. The control processor further includes an operating system portion having a kernel memory and at least one network driver communicating with the plurality of network interfaces. A buffer descriptor list is accessible by the network application and the network driver. The buffer descriptor list defines the status of buffers provided in the kernel memory that are used for temporary storage of data packets transferred between the network application and the plurality of network interfaces via the network driver. Data packets received by the network interfaces from the external networks directed to the network application are placed in selected ones of the buffers by the network driver for direct access by the network application. Similarly, data packets transmitted from the network application to the external networks are placed in other selected ones of the buffers for direct access by the network driver.

23 Claims, 9 Drawing Sheets



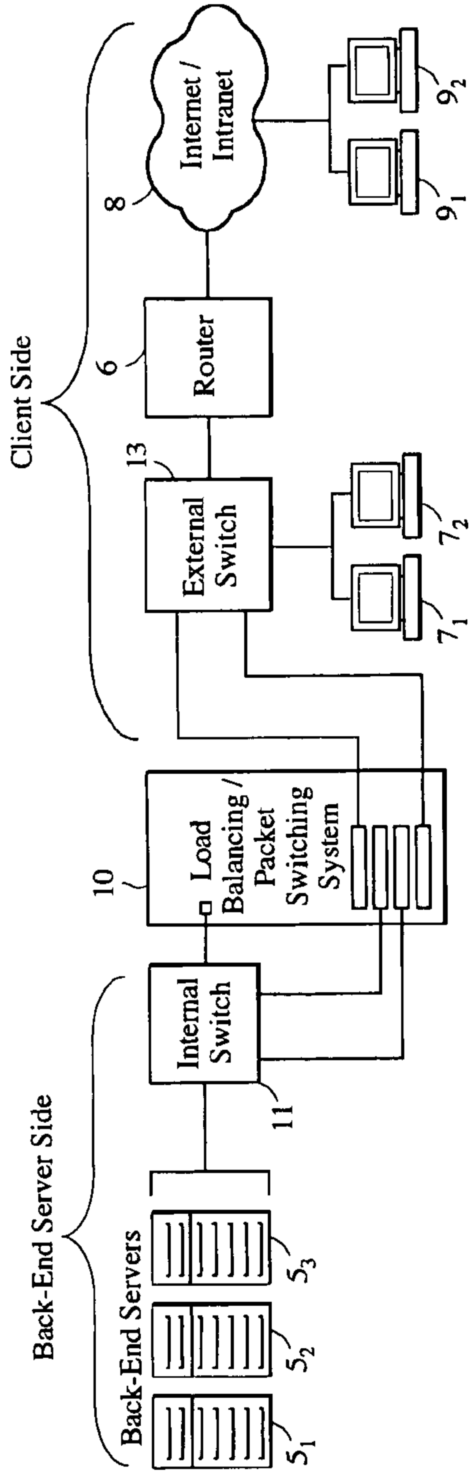

Client Side
Internet / Intranet
8
9_1
9_2
Router
6
External Switch
13
7_1
7_2
Load Balancing / Packet Switching System
10
Internal Switch
11
Back-End Server Side
Back-End Servers
5_1
5_2
5_3

FIG.1

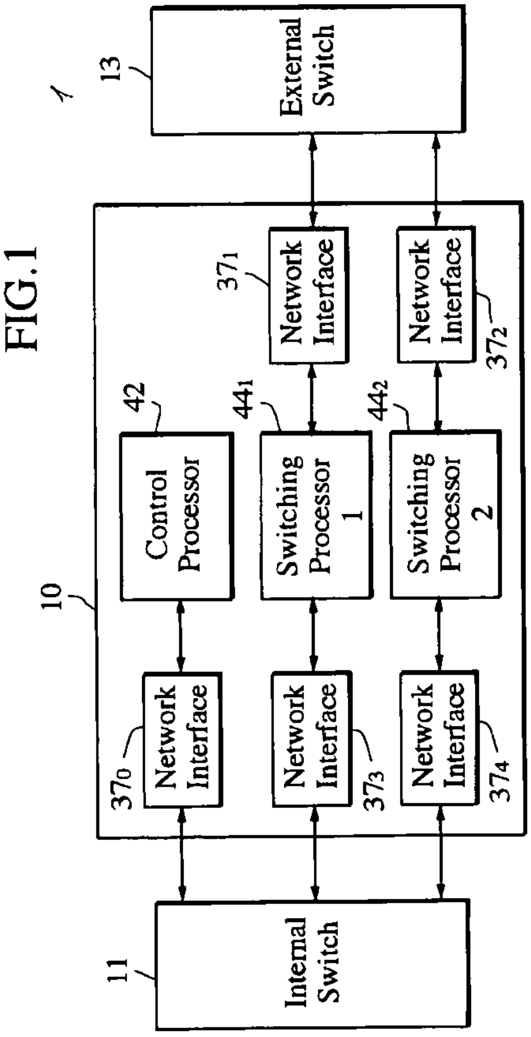

1
13
External Switch
10
Network Interface
37_1
Network Interface
37_2
42
Control Processor
44_1
Switching Processor 1
44_2
Switching Processor 2
Network Interface
37_0
Network Interface
37_3
Network Interface
37_4
11
Internal Switch

FIG.7

FIG. 2

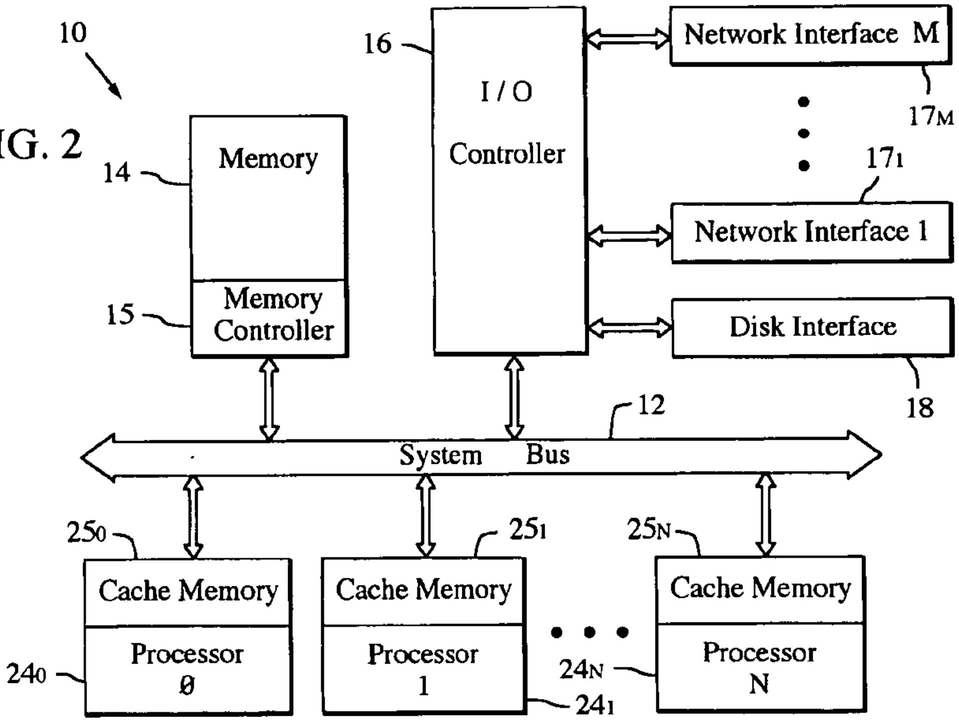
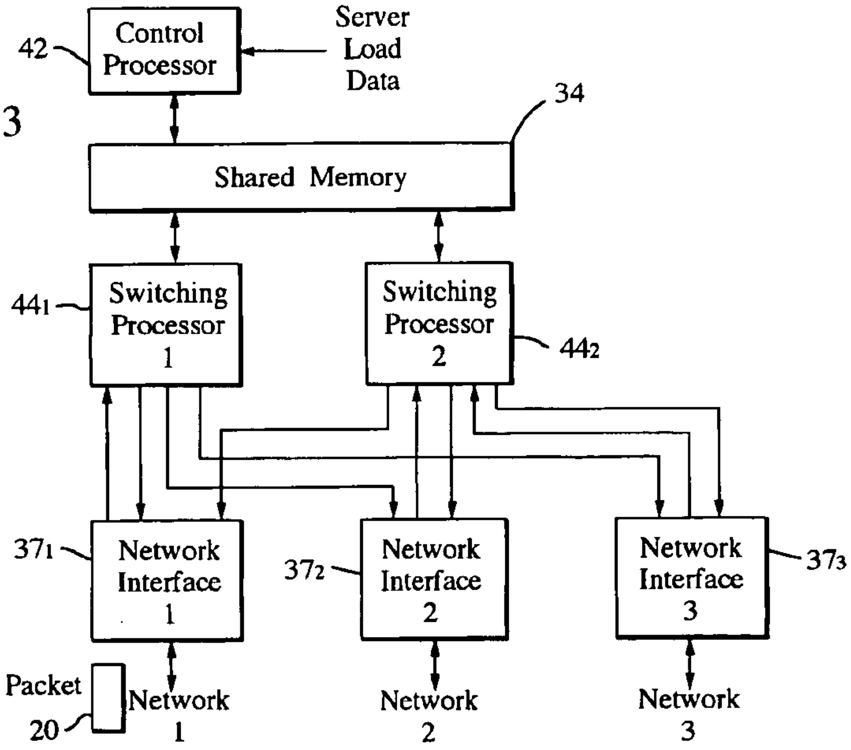

10
16
I / O
Controller
Network Interface M
17M
17₁
Network Interface 1
Disk Interface
18
14
Memory
15
Memory Controller
12
System Bus
25₀
25₁
25N
Cache Memory
Processor 0
Cache Memory
Processor 1
Cache Memory
Processor N
24₀
24N
24₁

FIG. 3


42
Control Processor
Server Load Data
34
Shared Memory
44₁
Switching Processor 1
Switching Processor 2
44₂
37₁
Network Interface 1
37₂
Network Interface 2
Network Interface 3
37₃
Packet
20
Network 1
Network 2
Network 3

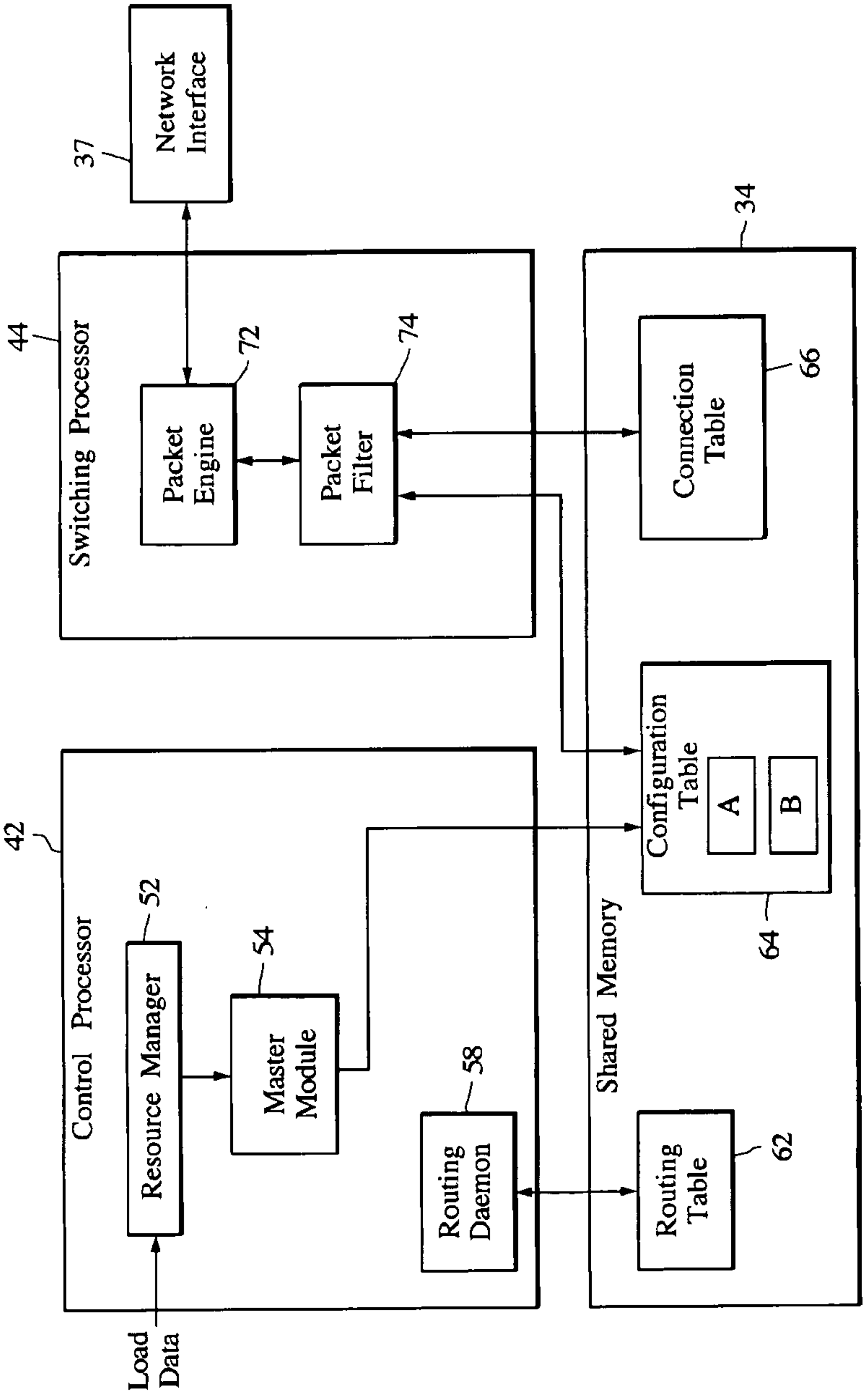
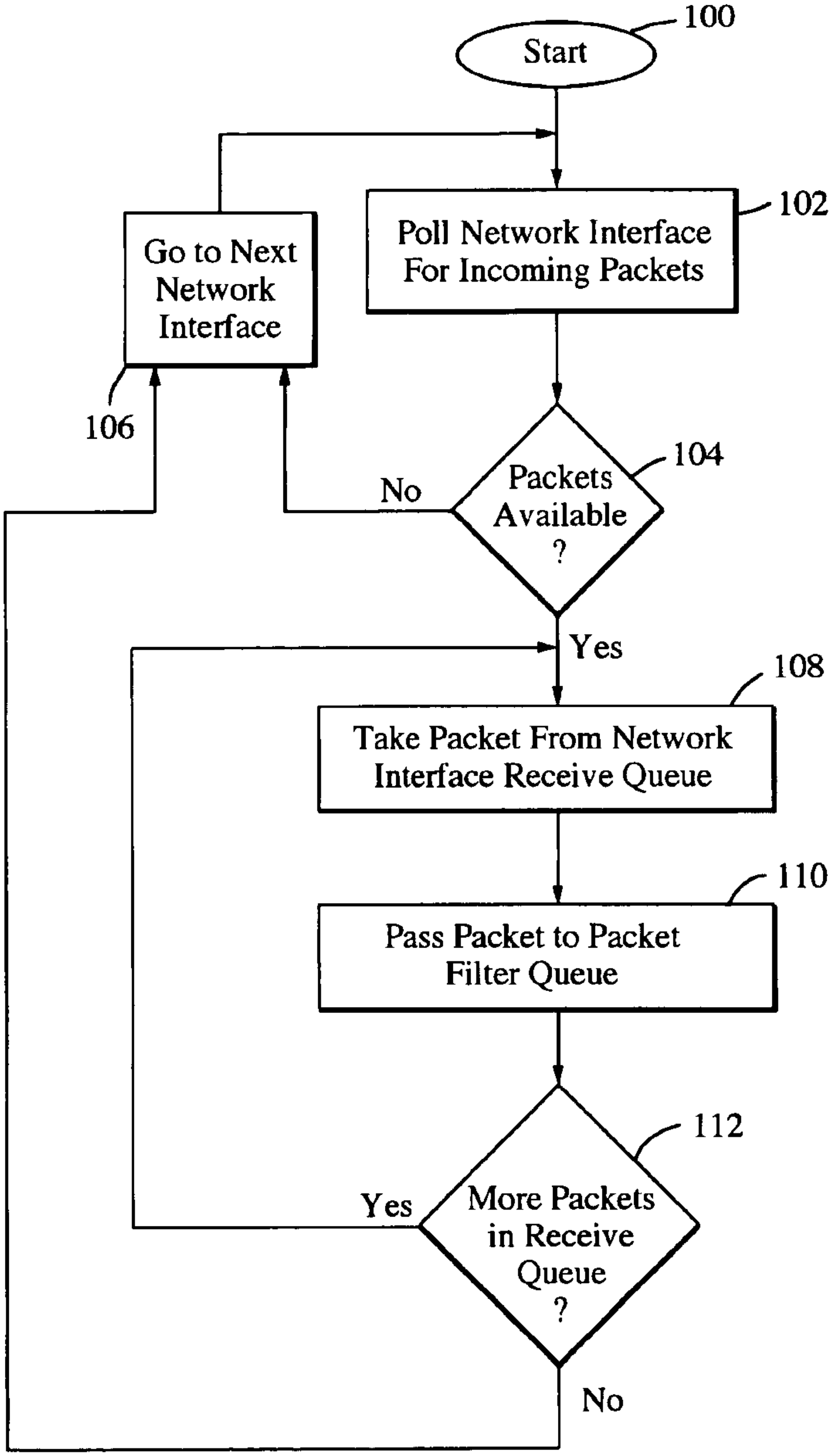

37
Network Interface
44
Switching Processor
72
Packet Engine
74
Packet Filter
34
Connection Table
66
42
Control Processor
52
Resource Manager
54
Master Module
58
Routing Daemon
Shared Memory
Configuration Table
A
B
64
Routing Table
62
Load Data

FIG. 4

FIG. 5


Start
100
Poll Network Interface For Incoming Packets
102
Go to Next Network Interface
106
Packets Available ?
104
No
Yes
Take Packet From Network Interface Receive Queue
108
Pass Packet to Packet Filter Queue
110
More Packets in Receive Queue ?
112
Yes
No

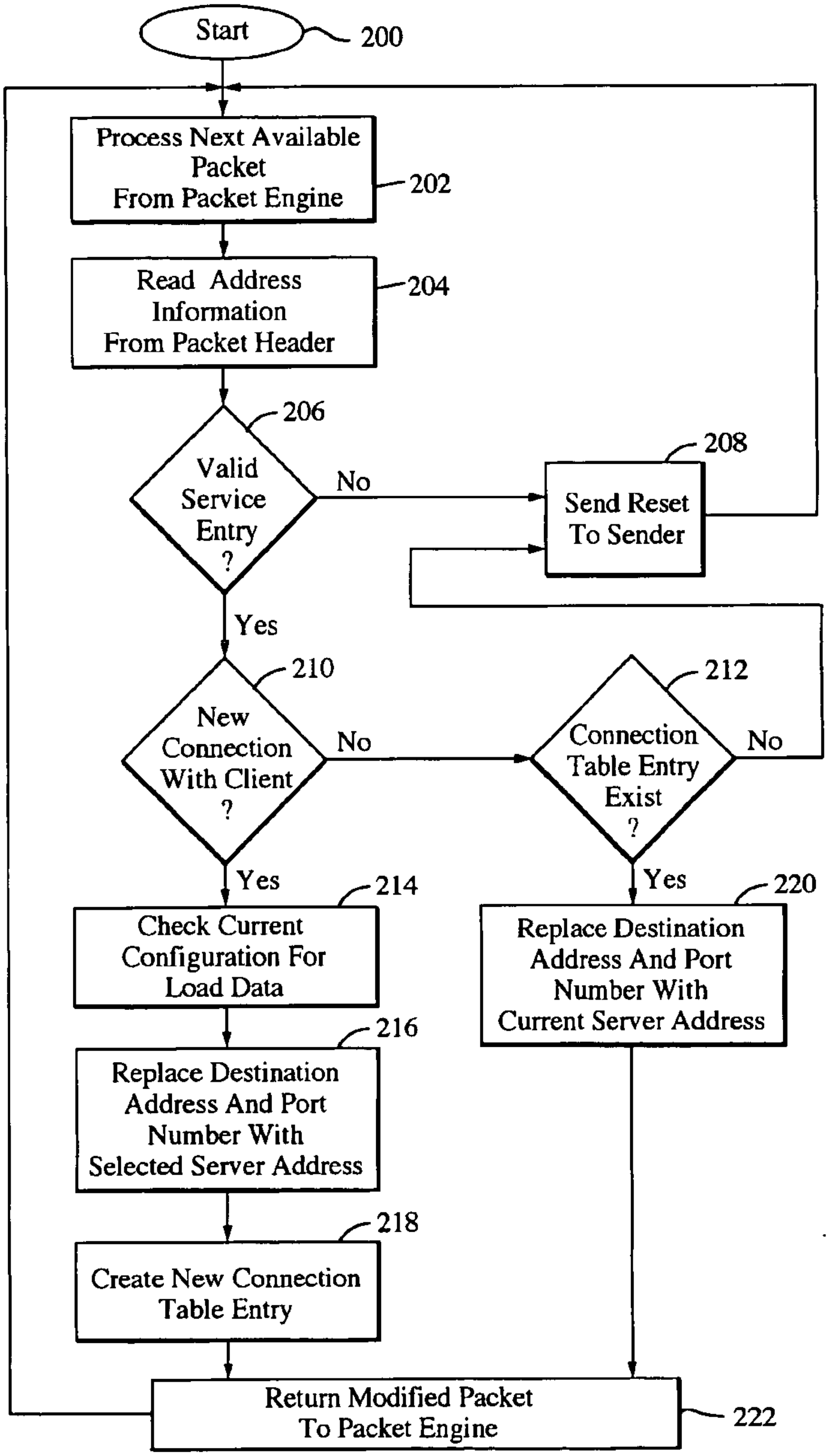

Start
200
Process Next Available Packet From Packet Engine
202
Read Address Information From Packet Header
204
206
Valid Service Entry ?
No
208
Send Reset To Sender
Yes
210
New Connection With Client ?
No
212
Connection Table Entry Exist ?
No
Yes
214
Check Current Configuration For Load Data
Yes
220
Replace Destination Address And Port Number With Current Server Address
216
Replace Destination Address And Port Number With Selected Server Address
218
Create New Connection Table Entry
Return Modified Packet To Packet Engine
222

FIG. 6

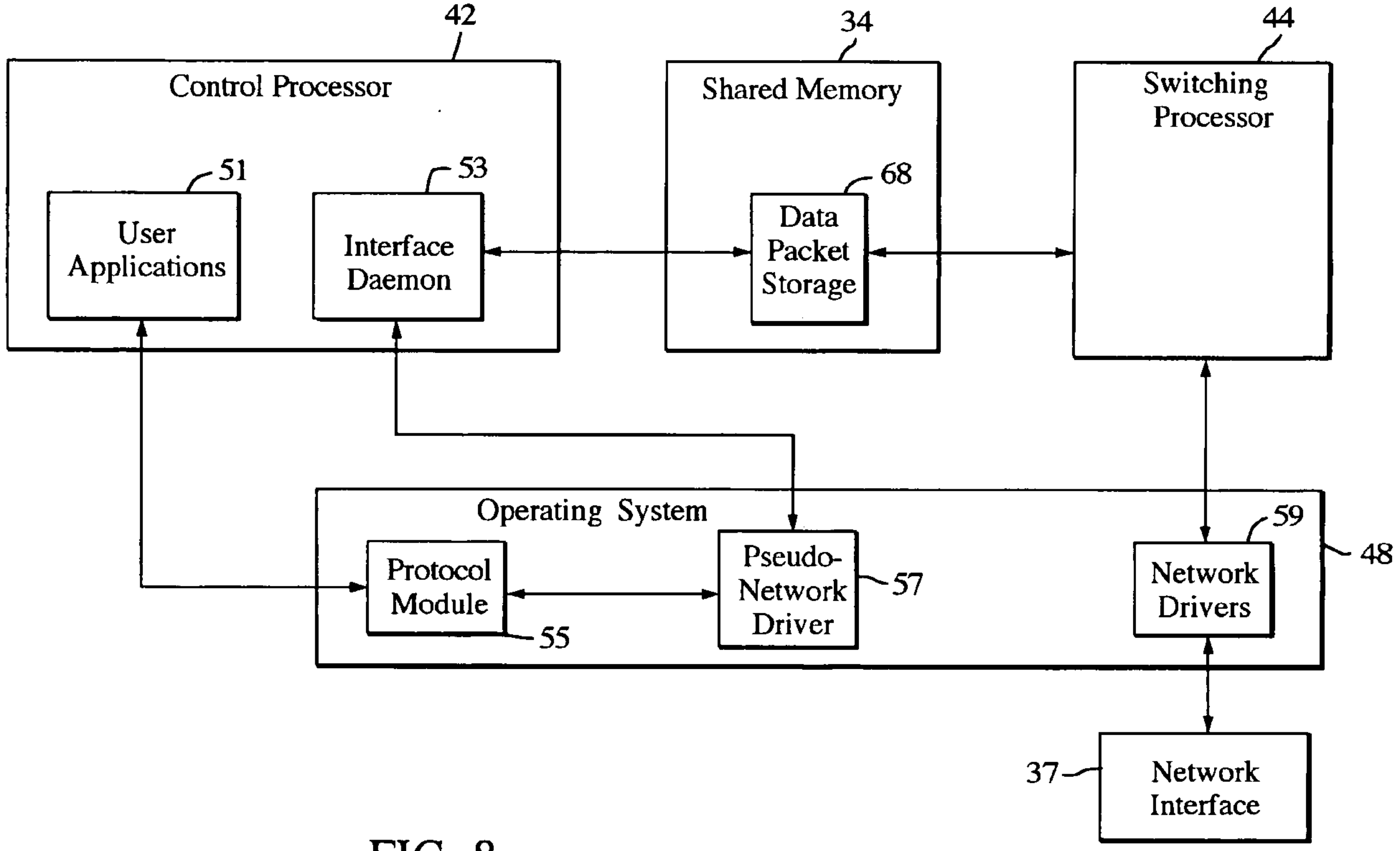

42
Control Processor
51
User Applications
53
Interface Daemon
34
Shared Memory
68
Data Packet Storage
44
Switching Processor
Operating System
Protocol Module
55
Pseudo-Network Driver
57
59
Network Drivers
48
37
Network Interface

FIG. 8

FIG.9

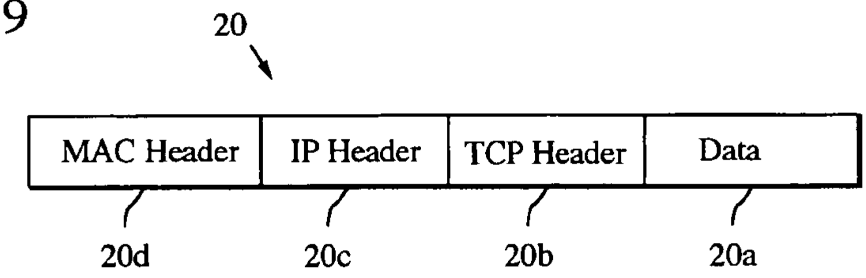
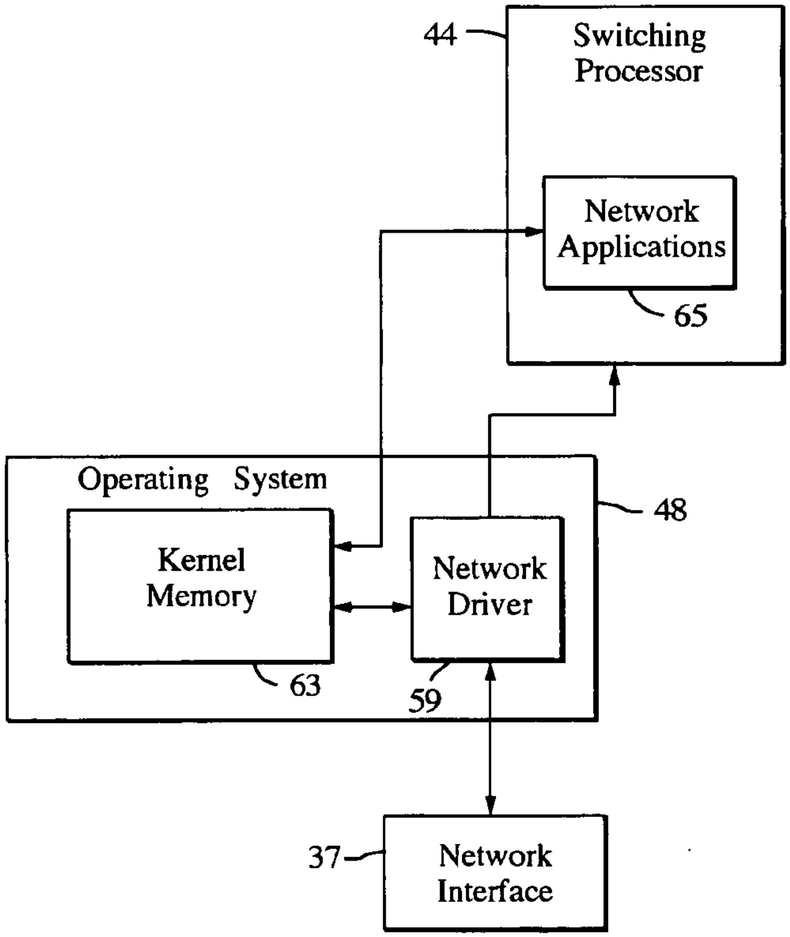

20
MAC Header
IP Header
TCP Header
Data
20d
20c
20b
20a

FIG.10


44
Switching Processor
Network Applications
65
Operating System
48
Kernel Memory
63
Network Driver
59
37
Network Interface

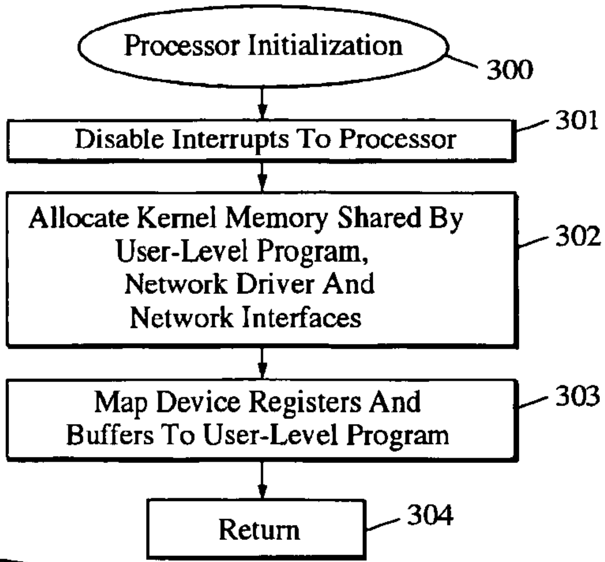

Processor Initialization
300
Disable Interrupts To Processor
301
Allocate Kernel Memory Shared By User-Level Program, Network Driver And Network Interfaces
302
Map Device Registers And Buffers To User-Level Program
303
Return
304

Fig. 11

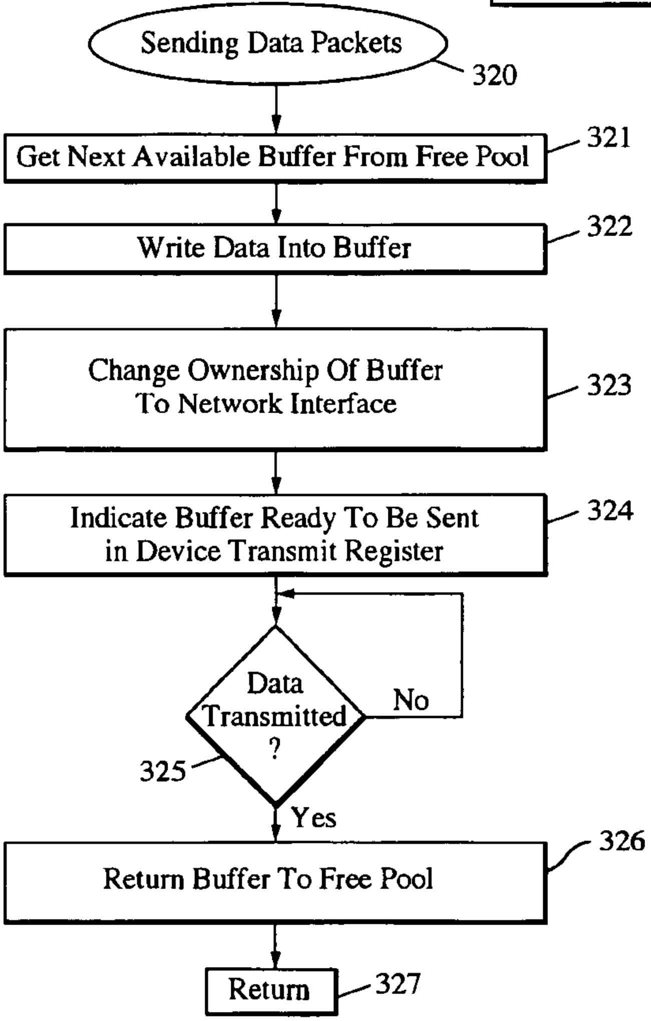
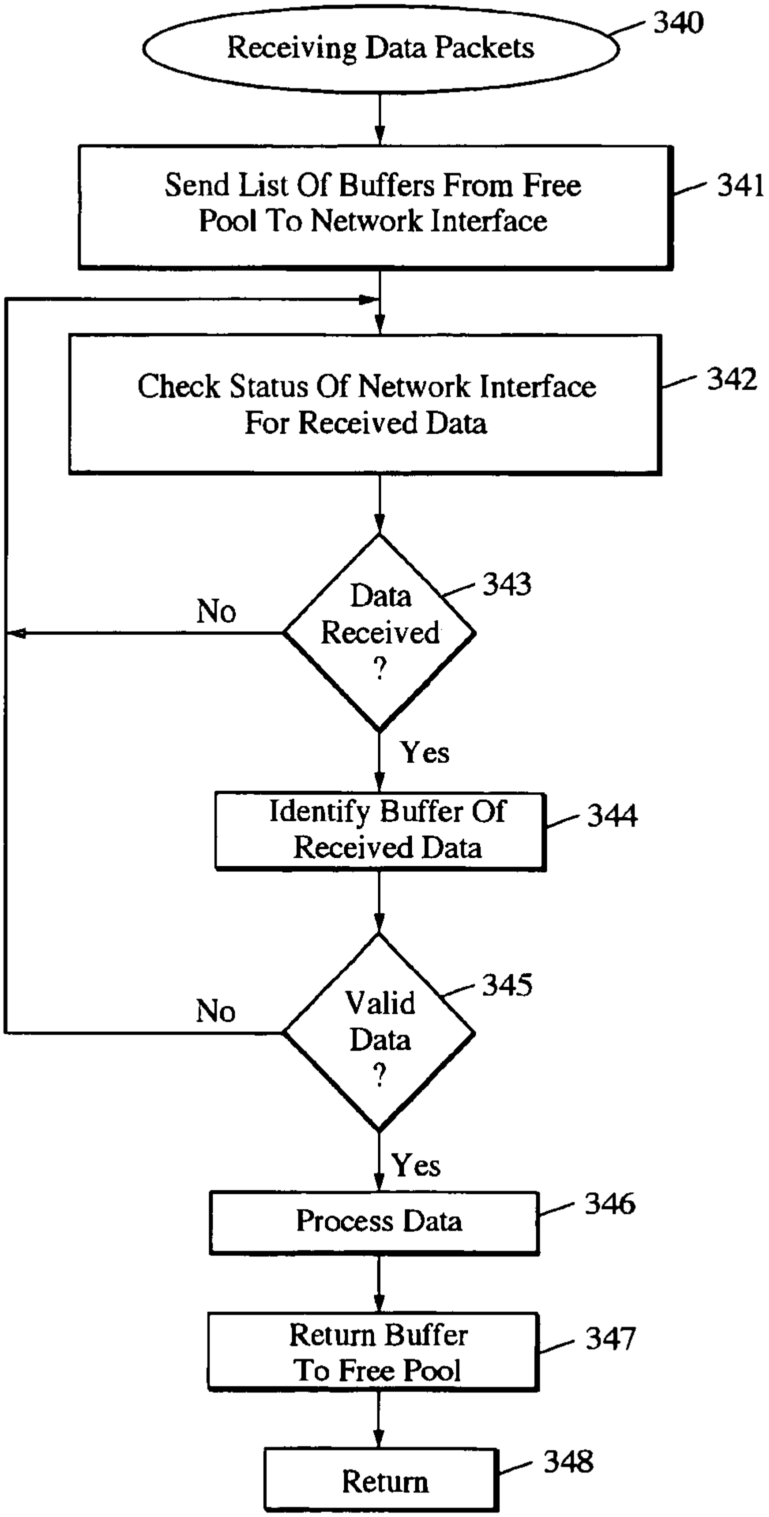

Sending Data Packets
320
Get Next Available Buffer From Free Pool
321
Write Data Into Buffer
322
Change Ownership Of Buffer To Network Interface
323
Indicate Buffer Ready To Be Sent in Device Transmit Register
324
Data Transmitted ?
No
325
Yes
Return Buffer To Free Pool
326
Return
327

Fig. 12

Fig. 13


Receiving Data Packets
340
Send List Of Buffers From Free Pool To Network Interface
341
Check Status Of Network Interface For Received Data
342
Data Received ?
343
No
Yes
Identify Buffer Of Received Data
344
Valid Data ?
345
No
Yes
Process Data
346
Return Buffer To Free Pool
347
Return
348

USER-LEVEL DEDICATED INTERFACE FOR IP APPLICATIONS IN A DATA PACKET SWITCHING AND LOAD BALANCING SYSTEM

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates generally to computer data communication networks, and more particularly, to a multiprocessor computer architecture having plural switching modules for transferring data packets between computer networks and a control module for performing load balancing to ensure efficient utilization of the computer networks in which a software interface is defined between the switching module and the operating system for transferring data packets therebetween.

2. Description of Related Art

Computer networks are widely used as a way to communicate messages between computers. The Internet is made up of more than 100,000 interconnected computer networks spread across over 100 countries, including commercial, academic and government networks. Originally developed for the military, the Internet has become widely used for academic and commercial research. Today, the Internet has become commercialized into a worldwide information highway, providing information on every subject known to humankind. Similarly, businesses and other entities have adopted the Internet paradigm as a model for their internal networks, or so-called "intranets."

Messages transferred between computers within a network are typically broken up into plural data packets. Packet switching systems are used to route the data packets to their required destination and enable the efficient handling of messages of different lengths and priorities. Since each data packet includes a destination address, all packets making up a single message do not have to travel the same path. Instead, the data packets can be dynamically routed over the interconnected networks as circuits become available or unavailable. The destination computer receives the data packets and reassembles them back into their proper sequence to reconstruct the transmitted message.

Internet computer networks generally use the TCP/IP communications protocol, which is an acronym for Transmission Control Protocol/Internet Protocol. The TCP portion of the protocol provides the transport function by breaking a message into smaller packets, reassembling the packets at the other end of the communication network, and re-sending any packets that get lost along the way. The IP portion of the protocol provides the routing function by giving the data packets an address for the destination network and client at the destination network. Each data packet communicated using the TCP/IP protocol includes a header portion that contains the TCP and IP information. Another communication protocol used in communication between Internet computer networks is UDP/IP, in which UDP is an acronym for User Datagram Protocol. UDP is used in place of TCP in conditions when a reliable delivery is not required. For example, UDP/IP is often used for real-time audio and video traffic where lost data packets are simply ignored, because there is no time to retransmit. Since the computer networks connected to the Internet may use other communication protocols besides TCP/IP or UDP/IP, gateways are used to convert data packets from these protocols into the other protocols.

At a destination network, one or more routers may be utilized to receive incoming data packets and route the packets to other internal networks such as local area networks (LAN). The internal networks may further include servers that supply information to one or more clients. The servers are generally high-speed microcomputers, minicomputers or even mainframes. In some cases, the clients are internal to the network (i.e., at the back-end), and the router acts as a conduit for communication of data packets between the clients and the outside world. The back-end servers may provide various application functions for the clients, such as a database server that maintains the databases and processes requests from clients to extract data from or update the databases. In other cases, the clients are external to the network (i.e., at the front-end), and the router acts as a conduit for communication of data packets between the clients and the back-end servers. For example, an Internet application server at the back-end may host Web applications within the network that are accessed by clients outside the network. In still other cases, the clients are both internal and external to the network. The routers perform the functions of switching data packets between the internal and external networks, and balancing the load placed upon the back-end servers of the internal network by distributing message packets between the back-end servers in the most efficient and expeditious manner.

In view of the high volume of message traffic that they process and the relatively limited kinds of tasks that they perform, routers typically comprise dedicated switching processors having an architecture optimized to provide these functions. These conventional dedicated switching processors include a control module and a switching module that are viewed by the external networks as a single network entity. A drawback of such dedicated switching processors is that they can be very expensive due in part because they are manufactured in relatively low volumes as compared with other general-purpose computer systems. Moreover, the software that provides the message routing and load balancing functions must be written specifically for the dedicated switching processors, which further increases the cost of purchasing, operating and maintaining such systems. An additional drawback of dedicated switching processors is that most modifications to their functionality require a hardware change, which is typically more expensive and difficult than a software change.

A further disadvantage of dedicated switching processors is that it is cumbersome to communicate data packets between the switching module and the control module. Generally, the control module communicates with the switching module through special internal interfaces that add overhead to both the control module and the switching module, and is thus undesirable. For example, the control module may include network applications that operate at the user level, and data input and output for the network applications is handled at the operating system level. The operating system communicates with the network devices and issues interrupts to the network applications at the user level to indicate the receipt of data. These conventional systems are inefficient since processing of the network applications is stopped each time an interrupt is issued, and the involvement of the operating system further reduces the efficiency of the network applications.

It would therefore be very desirable to provide the message routing and load balancing functions of a network router within a general-purpose symmetrical multiprocessor (SMP) computer system. Such general-purpose multiprocessor computer systems are less expensive than conventional systems due to their larger volume production, and changes to their functionality can be readily accomplished by modifying their software rather than their hardware. It would additionally be desirable to provide network applications operating on a general-purpose multiprocessor computer system direct access to the network interfaces and to run the network applications on a dedicated processor which is not interrupted. Certain applications such as Internet telephony or fax applications would particularly benefit from such direct network access.

SUMMARY OF THE INVENTION

In accordance with the teachings of the present invention, a data packet switching and server load balancing device is provided by a general-purpose multiprocessor computer system. The general-purpose multiprocessor computer system comprises a plurality of symmetrical processors coupled together by a common data bus, a main memory shared by the processors, and a plurality of network interfaces each adapted to be coupled to respective external networks for receiving and sending data packets via a particular communication protocol, such as Transmission Control Protocol/Internet Protocol (TCP/IP) or User Datagram Protocol (UDP).

More particularly, a first one of the processors is adapted to serve as a control processor and remaining ones of the processors are adapted to serve as data packet switching processors. The data packet switching processors are each coupled to at least one of the plurality of network interfaces. The control processor receives raw load status data from agents running on the back-end application servers and generates load distribution configuration data therefrom. The load distribution configuration data is stored in the main memory for access by the data packet switching processors. The switching processors route received ones of the data packets to a selected one of the external networks in accordance with information included in a header portion of the data packets and the load distribution configuration data. The switching processors perform periodic polling of corresponding ones of the network interfaces to detect a received one of the data packets therein. In addition, the switching processors re-write the routing information included in the header portion of the data packets to reflect the selected one of the external networks.

In an embodiment of the invention, a multiprocessor computer system comprises a plurality of network interfaces each adapted to be coupled to respective external networks for receiving and sending data packets to and from remote devices coupled to the external networks via a particular communication protocol. The multiprocessor computer system further comprises a plurality of symmetrical processors including a control processor and at least one switching processor. The switching processor further includes at least one network application executing thereon. The control processor further includes an operating system portion having a kernel memory and at least one network driver communicating with the plurality of network interfaces. A buffer descriptor list is accessible by the network application and the network driver. The buffer descriptor list defines the status of buffers provided in the kernel memory that are used for temporary storage of data packets transferred between the network application and the plurality of network interfaces via the network driver. Data packets received by the network interfaces from the external networks directed to the network application are placed in selected ones of the buffers by the network driver for direct access by the network application. Similarly, data packets transmitted from the network application to the external networks are placed in other selected ones of the buffers for direct access by the network driver.

A more complete understanding of the software interface between switching and control modules of a computer data packet switching and load balancing system will be afforded to those skilled in the art, as well as a realization of additional advantages and objects thereof, by a consideration of the following detailed description of the preferred embodiment. Reference will be made to the appended sheets of drawings, which will first be described briefly.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a block diagram of a network configuration having a load balancing and packet switching device in accordance with the present invention;

FIG. **2** is a block diagram of a general-purpose symmetrical multiprocessor computer system adapted to provide the load balancing a packet switching device;

FIG. **3** is a block diagram of the general-purpose multiprocessor computer system configured to provide a switching processor to perform network data packet switching and a control processor to perform network load balancing;

FIG. **4** is a block diagram depicting communication of information between the control processor and one of the switching processors;

FIG. **5** is a flow chart illustrating operation of the packet engine module of the switching processor;

FIG. **6** is a flow chart illustrating operation of the packet filter module of the switching processor;

FIG. **7** is a block diagram illustrating a first embodiment of the invention having a pseudo-interface between the control processor and switching processors through the internal switch;

FIG. **8** is a block diagram illustrating a second embodiment of the invention having a pseudo-interface between the control processor and switching processors through a driver operating on the control processor;

FIG. **9** is a block diagram illustrating the portions of a data packet;

FIG. **10** is a block diagram illustrating a third embodiment of the invention having a user-level network interface for applications operating on the switching processor;

FIG. **11** is a flow chart illustrating a process of initializing the switching processor for user-level access to the network interfaces;

FIG. **12** is a flow chart illustrating a process of sending data packets to a network interface at the user level; and

FIG. **13** is a flow chart illustrating a process of receiving data packets from a network interface at the user level.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

The present invention satisfies the need for a general-purpose multiprocessor computer system adapted to provide message routing and load balancing functions for a computer network. In the detailed description that follows, like element numerals are used to describe like elements depicted in one or more of the figures.

Referring first to FIG. **1**, an exemplary network configuration using a load balancing and packet switching system **10** of the present invention is illustrated. The network elements illustrated to the left of the load balancing and packet switching system **10** in FIG. **1** are referred to as the "back-end server" side of the network, and the network elements illustrated to the eight of the load balancing and packet switching system **10** are referred to as the "client" side of the network. On the client side of the network, the load balancing and packet switching system **10** is coupled through two separate network channels to an external network switch **13**. The external switch **13** is coupled to client stations 7_1–7_2, permitting communication between the client stations and the back-end server side of the network. The external switch **13** is further connected to the Internet (or an intranet) **8** servicing client stations 9_1–9_2 through a router **6**. On the back-end server side of the network, the load balancing and packet switching system **10** is coupled through two separate network channels to an internal network switch **11**. The internal switch **11** is further coupled to back-end servers 5_1–5_3. Thus, data packets originated at the client side of the network, such as from client stations 9_1–9_2 or 7_1–7_2, and directed to the back-end side of the network pass through the external switch **13** to the load balancing and packet switching system **10**, which determines the routing of the data packets to the back-end servers 5_1–5_2 through the internal switch **11**. Conversely, datar packets originated at the back-end side of the network and directed to the client side of the network follow the same path in reverse.

As known in the art, a network switch is a device that cross connects network nodes or LAN segments and allows full bandwidth to pass between connected nodes. Alternatively, the internal or external switches **11**, **13** could be provided by a network hub, which is a device that connects nodes by sharing the bandwidth between the connected nodes. Network switches are advantageous over network hubs in view of their greater capacity and speed. As also known in the art, a router is a device that routes data packets between networks. Routers read the network address in each transmitted data packet and make a decision on how to send it based on the most expedient route (traffic load, line costs, speed, bad lines, etc.). Alternatively, the router **6** may be provided by a network switch or hub. It should be appreciated that various alternative network configurations are anticipated, and moreover, that the numbers of clients, back-end servers and network channels shown in FIG. **1** are purely for the purpose of illustration and are not intended to limit the scope of the invention in any way.

Referring now to FIG. **2**, there is shown a general-purpose symmetrical multiprocessor (SMP) computer adapted to provide the load balancing and packet switching system **10** of FIG. **1**. The SMP computer includes N individual processors 24_0–24_N coupled to a common system bus **12**. Each one of the N processors 24_0–24_N has an associated cache memory $^{25}{}_0$–25_N. The processors 24_0–24_N may be provided by 64-bit UltraSPARC microprocessors sold by Sun Microsystems, Inc. The SMP computer further includes a main memory **14** and a memory controller **15** coupled to the common system bus **12**. The main memory **14** contains stored data and instructions accessible by each of the processors 24_0–24_N with the memory controller **15** controlling individual accesses to the main memory. As known in the art, the cache memory 25_0–25_N bridges the main memory **14** and the processors 24_0–24_N. The cache memory 25_0–25_N is faster than the main memory **14** and allows instructions to be executed and data to be read at higher speed. Instructions and data are transferred to the cache memory 25_0–25_N in blocks using a look-ahead algorithm. The more sequential the instructions in the routine being accessed, and the more sequential the order of the data being read, the more chance the next desired item will still be in the cache memory 25_0–25_N, and the greater improvement in performance. It is anticipated that the cache memory 25_0–25_N be comprised of static random access memory (SRAM) chips, while dynamic RAM (DRAM) chips are used for main memory **14**.

Alternatively, the cache memory 25_0–25_N may be provided directly onto the same chips as the corresponding processors 24_0–24_N.

An input/output (I/O) controller **16** is also coupled to the common system bus **12**, and controls the transfer of data between the processors 24_0–24_N and peripheral devices. In particular, the I/O controller **16** is coupled to a disk interface device **18** which controls exchanges of data between the processors 24_0–24_N and one or more disk storage devices. The I/O controller **16** is also coupled to M network interface devices 17_1–17_M which each control exchanges of data between the processors 24_0–24_N and external computer networks, clients or servers. Each one of the network interface devices 17_1–7_M include a receive queue in which received data packets are temporarily held while awaiting processing by the SMP computer, and a transmit queue in which transmitted data packets are temporarily held while awaiting communication to a computer network. It should be appreciated that the N number of processors 24_0–24_N would generally be equal to or less than the M number of network interface devices 17–17_M. Each of the M network interface devices 17_1–17_M may communicate with plural computer networks, clients or servers, using conventional network protocols such as Ethernet, Token Ring, Asynchronous Transfer Mode (ATM), etc.

It should be appreciated that the SMP computer may further include a keyboard and monitor (not shown) to permit access by management information services (MIS) personnel, such as to perform diagnostics, routine maintenance, and administrative level tasks. As will be further described below, the SMP computer is adapted to provide message routing and load balancing; functions that would not require any direct user interaction, and the keyboard and monitor would therefore serve little use during ordinary operation of the computer system. However, certain applications of the load balancing and message routing system do include user applications running on the SMP computer, and for such applications it should be appreciated that a keyboard and monitor would be necessary. It is anticipated that the SMP computer include a multitasking, multiprocessing operating system, such as the Solaris operating system by Sun Microsystems, Inc.

Referring now to FIG. **3**, a block diagram of the general-purpose SMP computer configured to provide network data packet switching and load balancing functions is illustrated. In the load balancing and packet switching system **10**, one of the plural processors 24_0–24_N of FIG. **1** serves as a control processor **42**, and the remaining processors serve as switching processors 44_1 and 44_2. The control processor **42** and switching processors 44_1 and 44_2 each have access to a shared memory space **34**, such as provided by a portion of the main memory **14** of FIG. **1**. The control processor **42** handles administrative and configuration functions for the load balancing and packet switching system **10**, and also communicates with agents on the application servers to collect system load information. The control processor **42** then performs complex calculations on the raw system load information and defines an optimum traffic load distribution. The traffic load distribution result is then written into the shared memory space for use by the switching processors 44_1 and 44_2. The switching processors 44_1 and 44_2 exclusively perform the packet switching tasks, and do not handle any other computing tasks. Although two switching processors 44_1 and 44_2 are depicted in FIG. **3**, it should be appreciated that any number of switching processors can be advantageously utilized.

The switching processors 44_1 and 44_2 are each coupled to plural network interfaces 37_1–37_3, such as provided by the network interfaces 17_1–17_M of FIG. 2. Each respective one of the switching processors 44_1 and 44_2 poll corresponding ones of the plural network interfaces 37_1–37_3 for incoming data packets 20 present on their respective receive queues. Particularly, switching processor 44_1 polls the receive queue of network interface 37_1, and switching processor 44_2 polls the receive queue of network interfaces 37_2 and 37_3. Since; each of the switching processors 44_1 and 44_2 poll different ones of the network interfaces 37_1–37_3, conflicts between the switching processors over received data packets is avoided. In contrast, each one of the switching processors 44_1 and 44_2 can supply data packets to the transmit queues of each one of the network interfaces 37_1–37_3, so that data packets can then be routed to any computer network coupled to the load balancing and packet switching system 10.

FIG. 4 illustrates in greater detail the communication of information between the control processor 42 and one of the switching processors 44. The control processor 42 further includes several software modules to handle discrete control tasks, including a resource manager module 52 and a master module 54. The control processor 42 may further include specialized application program interfaces (API) that handle communication between these software modules. The resource manager module 52 receives raw data from the back-end application servers indicating their present load status. This raw data includes various factors, including the number of clients presently being served, the utilization rates of the CPU and memory of the application server processor, the average execution time, and the number of requests per second. The raw load data is then provided to the master module 54, which synthesizes the data into a desired load distribution in accordance with a predetermined distribution algorithm. For example, the distribution algorithm may favor distribution of incoming packets so that all servers have an even load, or alternatively, may favor distribution of incoming packets to certain servers having unique applications or processing capability. Such distribution algorithms are well known in the art. It is also anticipated that the resource manager module 52 can be provided as a separate device entirely external to the control processor 42.

The shared memory 34 further includes a routing table 62, a configuration table 64, and a connection table 66. The routing table 62 is a database that contains the current network topology, and is accessed by the switching processor 44 in determining routing information for the received data packets. Specifically, the routing table 62 defines the addresses and interconnection pathways between the load balancing and packet switching device 10 and the networks connected thereto. A routing daemon 58 within the control processor 42 is a program that executes in the background to retrieve the information stored in the routing table 62 and maintains the status of the routing table 62 as changes are made to the configuration. As generally known in the art, the routing daemon 58 functions like an extension to the operating system, and does not otherwise interact with the other modules of the control processor 42 or the switching processor 44 discussed above.

The load distribution data synthesized by the master module 54 is stored in the configuration table 64. The configuration table includes two redundant memory buffers, identified in FIG. 4 as A and B. At any given time, one of the two memory buffers is the active buffer and the other is the back-up buffer. A memory pointer within the shared memory 34 defines which one of the two buffers is currently the active buffer. The switching processor 44 obtains the current load distribution data from the active buffer. The master

module 54 of the control processor 42 periodically provides updated load distribution data to the shared memory 34 that is written to the back-up buffer. Thereafter, the memory pointer switches from the active to the back-up buffer so that the updated load distribution data is accessible to the switching processor 44. This double buffering technique speeds up operation of the load balancing and packet switching system 10 by enabling load data processing to occur concurrently with packet switching, and prevents potential conflicts between the switching processor 44 and the control processor 42 that both need access to the same memory space.

The connection table 66 maintains a record of the TCP and UDP connections routed by each of the switching processors 44. As discussed above, the data packets received by the switching processors 44 each contain transport data in the header (i.e., TCP or UDP data) that defines how the data packets should be reassembled with other data packets to reconstruct complete messages, or connections. As shown in FIG. 9, the data packets 20 generally have an IP address which is provided in an IP header 20*c* to define the destination device as known to the external computer networks. This external IP address may actually be different than the internal IP address of the back-end application server selected by the load balancing and packet switching system 10. Accordingly, the entries of the connection table 66 map the external IP address to the internal IP address. Following the IP header 20*c*, a TCP (or UDP) header 20*b* contains the transport data. The data portion 20*a* of the data packet 20 is provided after each of the foregoing headers. Returning now to FIG. 4, a new entry is added to the connection table 66 after a first data packet of a new connection is received. The transport data for each of the received data packets is provided to the connection table 66 by the switching processor 44.

Once the IP address is translated by the connection table 66, the switching processor 44 determines a Media Access Control (MAC) address using an address resolution protocol (ARP). According to the ARP, a remote network node desiring to transmit a data packet to another node transmits an ARP broadcast packet that is received by every node connected to the network. The receiving node responds with an ARP response packet that contains the MAC address of the receiving node. Thereafter, the remote network node uses the MAC address in a MAC header 20*d* of subsequent data packets. The remote network node then saves the MAC address in the ARP cache memory so that it won't need to send another ARP broadcast packet again.

Like the control processor, the switching processor 44 also includes software modules to handle discrete tasks, including a packet engine module 72 and a packet filter module 74. The packet engine module 72 communicates with the network interface 37 to periodically poll for the presence of data packets in the receive queue, and delivers packets to the transmit queue to be sent to the external networks. The packet filter module 74 reads the IP and TCP/UDP data in the packet header to determine how to route the data packet. The packet filter module 74 accesses the connection table 66 in the shared memory 34 to determine whether a received packet is part of an existing connection or a new connection. Then, the packet filter module 74 accesses the configuration table 64 to determine the proper routing of the received data packet based on current load conditions and other factors. The switching processor 44 may further include specialized APIs that handle communication between these software modules.

The flow chart of FIG. 5 illustrates the software process performed by the packet engine module 72 of FIG. 4. The software process operates in a tight loop so as to minimize the latency of data packets waiting in the network interface receive queue. The process is initialized at step **100**, during which the switching processor **44** performs certain start-up tasks, including checking the routing table **62** in the shared memory **34**. It is anticipated that the load balancing and packet switching device **10** remain continuously in an operational state, and so this initialization step may only be executed rarely.

A first processing loop begins with step **102**, in which the packet engine module **72** polls the network interface **37** receive queue. At step **104**, the packet engine module **72** determines whether there are any data packets available at the receive queue. If no data packets are available, the packet engine module **72** changes to the next network interface **37** at step **106**. As discussed above, a single switching processor **44** may be responsible for receiving incoming data packets from plural ones of the network interfaces. It should be appreciated, however, that if the switching processor **44** only has responsibility for one network interface **37**, then this step **106** may be bypassed. After step **106**, the packet engine module **72** returns to step **102**. This first processing loop will repeat indefinitely until a received data packet is detected at step **104**. If a data packet is available in the network interface receive queue, a second processing loop begins at step **108** at which the packet engine module **72** retrieves the data packet. Then, at step **110**, the retrieved data packet is passed to the packet filter module **74** for routing (described below). Thereafter, at step **112**, the packet engine module **72** determines whether additional packets are present at the network interface receive queue. If additional packets are present, the packet engine module **72** returns to step **108** and the second processing loop is repeated. If no additional packets are present, the packet engine module **72** returns to step **106** and the next network interface is polled.

The flow chart of FIG. **6** illustrates the software process performed by the packet filter module **74** of FIG. **4**. The process is initialized at step **200**, during which the switching processor **44** performs certain start-up tasks as in step **100** discussed above. At step **202**, the packet filter module **74** begins processing of a data packet retrieved by the packet engine module **72** as discussed above. The packet filter module **74** reads the TCP/IP or UDP data from the header of the data packet in step **204**. The TCP/IP or UDP data will determine the subsequent processing and routing of the data packet. At step **206**, the packet filter module **74** determines from the TCP/IP or UDP data whether the data packet is a valid service entry. In other words, the packet filter module **74** verifies that the data packet was properly routed to the load balancing and packet switching device **10**, or whether it was routed improperly and received by the network interface in error. If the data packet is not a valid service entry, at step **208**, the packet filter module **74** sends a TCP reset packet back to the originator of a TCP connection via the packet engine module **72** and the network interfaces, or simply discards the data packet of a UDP connection.

Assuming that the data packet is a valid service entry, the packet filter module **74** determines at step **210** whether the data packet is a new connection with a client. The packet filter module **74** checks the transport data in the data packet header against the entries in the connection table **66** in the shared memory **34** to determine whether previous data packets have been received from the same client previously. If it is a new connection, then the packet filter module **74** checks the configuration table **64** for the current load conditions to determine the routing of data packet. As discussed above, the packet filter module **74** may elect to send the data packet to the application server having the lightest current load. Alternatively, the packet filter module **74** may send the data packet to a certain one of the application servers based on particular aspects of the data packet, e.g., the data packet is part of a connection requiring processing capability unique to one of the application servers, or the data packet specifically requests action by a particular application server.

Once the packet filter module **74** determines which application server should receive the data packet, the packet filter module at step **216** re-writes the MAC address and optionally re-writes the IP address and TCP/UDP port number in the header of the data packet to reflect the address of the selected application server. Then, at step **218**, a new entry is made in the connection table **66** to reflect the new connection. The packet filter module **74** then returns the modified data packet back to the packet engine module **72** at step **224** for forwarding to the appropriate network interface **37**. The packet filter module **74** then returns to step **202** to process the next available data packet.

If it was determined at step **210** that the received data packet was not a new connection with the client, the packet filter module **74** determines at step **212** whether a corresponding entry in the connection table **66** exists. If there is no corresponding entry, a reset packet is sent for TCP connections or the packet is discarded for UDP connections at step **208**. Conversely, if the connection table **66** has a corresponding entry for the data packet, then, at step **220**, the packet filter module **74** re-writes the MAC address and optionally re-writes the IP address and TCP/UDP port number to reflect the application server and application that is already servicing the connection. The packet filter module **74** then returns the modified data packet back to the packet engine module **72** at step **224** for forwarding to the appropriate network interface **37**. The packet filter module **74** then returns to step **202** to process the next available data packet.

Conventional dedicated switching processors include a control module and a switching module that are viewed by the external networks as a single network entity. The control module communicates with the switching module through special internal interfaces that add overhead to both the control module and the switching modules, and is thus undesirable. An advantage of the load balancing and packet switching system **10** of the present invention is that the control processor **42** and the switching processors **44**$_1$–**44**$_2$ may be viewed as entirely separate logical networking end points even though they both reside within a single physical device. Therefore, external clients may communicate with applications running on the control processor **42** by sending data packets through the switching processors **44**$_1$–**44**$_2$, which, in turn, route the data packets to the control processor. The control processor **42** reverses the order to send data packets back to the external clients.

A first alternative embodiment of the invention is provided in FIG. **7**, which illustrates a block diagram of a pseudo-interface between the control processor **42** and the switching processors **44**$_1$–**44**$_2$. As discussed above with respect to FIG. **1**, the load balancing and packet switching device **10** communicates on the client side through an external switch **13** and on the back-end server side through an internal switch **11**. More particularly, the switching processor **44**$_1$ communicates with the external switch **13** through the network interface **37**$_1$, the switching processor **44**$_2$ communicates with the external switch **13** through the network interface **37**$_2$. Similarly, the switching processor **44**$_1$ communicates with the internal switch **11** through the network interface **37**$_3$, and the switching processor **44**$_2$

communicates with the internal switch 11 through the network interface 37_4. The control processor 42 also communicates with the internal switch 11 through the network interface 37_0.

A virtual IP address is assigned to the network interface 37_0. When external devices on the client side of the network wish to communicate with the control processor 42, a data packet is transmitted through the external switch 13 to one of the switching processors 44_1–44_2, with the IP header 20*c* of the data packet listing the virtual IP address as the destination. The switching processor 44 then processes the incoming data packet in the manner described above with respect to FIGS. 5 and 6. Specifically, the packet filter module 74 of the switching processor re-writes the IP header 20*c* of the data packet to reflect the real IP address of the network interface 37_0. The packet engine module 72 then routes the modified data packet to the internal switch 11 through a corresponding one of the network interfaces 37. The internal switch 11 then sends the modified data packet to the network interface 37_0 which then delivers the data packet to the control processor 42. The process is reversed for responses sent by the control processor 42 back to the external device that originated the connection. The control processor 42 sends a data packet via the network interface 37_0 having the real IP address through the internal switch 11 to one of the switching processors 44. The switching processor 44 re-writes the IP address to the virtual IP address known to the external device. The modified data packet is then sent out by the switching processor 44 through the external switch 13.

A second alternative embodiment of the invention is provided in FIG. 8, which illustrates a block diagram of a pseudo-interface between the control processor 42 and a switching processor 44. The control processor 42 actually operates at two levels in a time-shared manner, referred to as a user level and an operating system level. The user level comprises the systems accessible to the user, and may include one or more user application programs 51 executing thereon, such as an e-mail program, a server application, and/or an Internet browser. The resource manager 52 and master module 54 described above with respect to FIG. 4 also execute in the user level. The operating system level, also known as the kernel, provides the basic services for the control processor 42 as well as the switching processor 44, such as activating the hardware directly or interfacing to another software layer that drives the hardware.

As shown in FIG. 8, the operating system 48 further includes a protocol module 55, a pseudo-network driver 57, and a network driver 59. The protocol module 55 serves as a data interface for the user application programs 51. The protocol module 55 converts received data packets that are directed to one of the user application programs 51 from the TCP/IP or UDP/IP protocols into a format usable by the user application programs. Specifically, the protocol module 55 strips off the MAC header 20*d*, IP header 20*c*, and TCP header 20*b*, leaving the data portion 20*a* of the data packet 20 (see FIG. 9). The data portion 20*a* is then provided to the user application programs 51. Conversely, the protocol module 55 formats data sent out from the user application programs 51 into data packets in accordance with the TCP/IP or UDP/IP protocols, by adding the MAC header 20*d*, IP header 20*c*, and TCP (or UDP) header 20*b*.

The network drivers 59 provide an interface between the hardware network interfaces 37 and the software switching processor 44. As illustrated in FIG. 8, the control processor 42 does not have a direct connection to the network drivers 59. Instead, the pseudo-network driver 57 is configured to appear to the user application programs 51 as a hardware network interface. The pseudo-network driver 57 may be provided by a STREAMS mechanism, which is a feature of a UNIX-based system that provides a standard way of dynamically building and passing messages up and down a message stack. Ordinarily, messages from a user application are passed "downstream" to the network driver at the end of the stack, and messages from the network driver are passed "upstream" to the user application. In the present invention, the pseudo-network driver 57 provides a message stack that is accessed through the use of system calls issued by the user application programs 51 to communicate with remote devices through the pseudo-network driver 57. As will be further described below, a data packet storage area 68 within the shared memory 34 appears to the user application programs 51 as such a remote device.

The interface daemon 53 is a program that executes in the background in the user level of the control processor 42 to communicate with the switching processor 44 and the pseudo-network driver 57 to initiate transfers of data packets therebetween. As described above with respect to FIGS. 5 and 6, the switching processors 44 receive incoming data packets from remote devices through the network interfaces 37. At step 204 of FIG. 6, the packet filter module 74 reads the MAC address and IP information from the header of a received data packet in order to determine routing of the data packet. If the packet switching processor 44 determines at step 204 that the intended destination for the data packet is one of the user applications 51 running on the control processor 42, the data packet is written into the data packet storage location 68 of the shared memory 34. The switching processor 44 then signals the interface daemon 53 of the availability of the data packet. The interface daemon 53 moves the received data packet to the pseudo-network driver 57. The received data packet is then processed through the protocol module 55 as if it were an incoming data packet received through an actual network interface.

To send data packets that originate in one of the user applications 51 to a remote device, the foregoing process is reversed. More particularly, data packets from the user applications 51 are passed to the pseudo-network driver 57, and the interface daemon 53 monitors the pseudo-network driver for data packets. Once a data packet arrives at the pseudo-network driver from the user application 51, the interface daemon 53 reads the data packet and places it in the data packet storage location 68 of the shared memory 34. Then, the interface daemon 53 signals the switching processor 44 of the availability of the data packet in the data packet storage location 68. The switching processor 44 then retrieves the data packet from the shared memory 34, and routes the data packet to one of the network interfaces 37 in the same manner as described above. As a result, remote devices can communicate with user applications 51 running on the control processor 42 even though the control processor does not have a direct connection to a network interface. The user applications 51 executing on the control processor 42 think they are communicating directly with actual network interfaces.

As discussed above, user applications ordinarily operate at the user level, and data input and output is handled at the operating system level. The operating system communicates with the network devices and issues interrupts to the network applications at the user level to indicate the receipt of data. These conventional systems are inefficient since processing of the network applications is stopped each time an interrupt is issued, and the involvement of the operating system further reduces the efficiency of the user applications. It would therefore be desirable to give the network applications direct access to the network interfaces and to run the network applications on a dedicated processor which is not interrupted. Certain network applications such as Internet telephony or fax applications would particularly benefit from such direct network access.

A third embodiment of the invention is provided at FIG. 10, which illustrates a block diagram of a user-level network interface for applications running on the switching processor 44. The user-level network interface overcomes the inefficiencies of the conventional systems discussed above. In FIG. 10, the switching processor 44 has certain network applications 65 running thereon, including the packet switching functions described above. The network applications 65 and the packet switching program have direct access to a list of buffers in the kernel memory 63. In an Ethernet network, each network interface 37 has a list of buffers associated with it. These buffers can be used to transmit data as well as receive data. A network driver 59 on the operating system 48 communicates with the network interface 37 in the manner described previously, and also has access to the buffer list in the kernel memory 63.

In particular, the buffer list includes descriptors that identify the address of each buffer within the kernel memory 63, the length of the data stored in the buffer, and an ownership identification of the buffer (i.e., whether the buffer is presently "owned" or controlled by the network interface hardware or the network application software). The network interface 37 circles through the buffer list in the kernel memory 63 to access the buffers in order to send or receive data as necessary. Similarly, the network applications 65 on the switching processor 44 circle through the list of buffers to process the data. If the network interface 37 transmits data from a particular buffer, the network applications 65 reclaim the buffer and return it to a free buffer pool. Conversely, if the network interface 37 has just received data and placed the data in a particular buffer, the network applications 65 process the data.

FIGS. 11–13 illustrate the processes performed by the switching processor 44 to initiate the direct user access to the network interfaces, to send data packets to the network interfaces, and to receive data packets from the network interfaces. As shown in FIG. 11, the switching processor 44 is initiated in a process beginning at step 300. At step 301, all interrupts to the switching processor 44 are disabled so that the switching program and any network application programs are run exclusively on the processor. Any interrupts from any device are thereafter delivered to the control processor 42. Next, at step 302, the kernel memory 63 that is to be shared between the network interfaces 37 and the network applications 65 operating on the switching processor 44 is allocated. All the buffers within the kernel memory 63 are mapped to all of the network interfaces 37 so that any buffer can be used to transmit or receive data through any of the network interfaces. Lastly, at step 303, the network interfaces' registers and buffers are mapped to the network applications 65. This enables the network applications 65 to directly control the network interfaces 37 by changing the content of the registers and to perform read/write operations from/to the buffers directly.

Once the switching processor 44 is initiated in this manner, all data accesses from/to the network interfaces operate like conventional memory read/write operations by the network applications. High efficiency results from the fact that the network applications 65 and the switching program run on a single thread on a dedicated, non-interruptable processor. Also, there is no context switching since the programs running on the switching processor 44 are isolated as a separate group that is not available to any other processes or threads in the multiprocessor system.

The process of sending data from one of the network applications 65 to the network interface 37 is illustrated in FIG. 12, and begins with step 320. At step 321, the network application 65 gets the next available buffer from the free buffer pool. The free buffer pool may be maintained as a table within the kernel memory 63. The network application 65 then writes the data to be transmitted in the form of a data packet into the identified buffer at step 322, and changes the "ownership" of the buffer to the network interface 37 at step 323. At step 324, the network application 65 indicates to the network interface 37 that a buffer contains data ready to be transmitted. At step 325, the network application 65 periodically checks to see if the data has been transmitted. Once the data has been transmitted, the network application 65 returns the buffer to the free pool at step 326. At step 327, the network application 65 returns to performing other tasks.

The process of receiving data from the network interface 37 to one of the network applications 65 is illustrated in FIG. 13, and begins with step 340. At step 341, the network application 65 passes a list of available buffers from the free buffer pool to the network interface 37. At step 342, the network application 65 checks the status of the network interface 37 to see if data has been received. If no data has been received, step 343 causes the program to loop back and repeat step 342. If data has been received by the network interface 37, the network application 65 identifies the buffer into which the data has been received by checking the ownership bit at step 344. The network application 65 next verifies that valid data was received into the buffer at step 345, and if the data is not valid then the program returns to step 342. Conversely, if the received data is valid, then the network application 65 processes the data at step 346. Thereafter, the network application 65 returns the buffer to the free buffer pool at step 347. At step 348, the network application 65 returns to performing other tasks.

Having thus described a preferred embodiment of a computer data packet switching and load balancing system using a general-purpose symmetrical multiprocessor architecture, it should be apparent to those skilled in the art that certain advantages of the aforementioned system have been achieved. It should also be appreciated that various modifications, adaptations, and alternative embodiments thereof may be made within the scope and spirit of the present invention. The invention is further defined by the following claims.

What is claimed is:

1. A multiprocessor computer system, comprising:

a plurality of network interfaces each adapted to be coupled to respective external networks for receiving and sending data packets to and from remote devices coupled to said external networks via a particular communication protocol;

a plurality of symmetrical processors including a control processor and at least one switching processor, said at least one switching processor further including at least one network application executing thereon, said control processor further including an operating system portion that includes a kernel memory and at least one network driver communicating with said plurality of network interfaces; and

a buffer descriptor list accessible by said at least one network application and said at least one network driver, said buffer descriptor list defining status of buffers provided in said kernel memory used for temporary storage of data packets transferred between said at least one network application and said plurality of network interfaces via said network driver;

wherein, data packets received by said network interfaces from said external networks directed to said at least one network application are placed in selected ones of said buffers by said network driver for direct access by said at least one network application, and data packets transmitted from said at least one network application to said external networks are placed in other selected ones of said buffers for direct access by said network driver.

2. The multiprocessor computer system of claim 1, wherein said at least one switching processor further includes stored instructions to be executed by said at least one switching processor, said stored instructions comprising the steps of:

disabling interrupts directed to said at least one switching processor; and

circling through said buffer descriptor list to monitor ownership status of said buffers.

3. The multiprocessor computer system of claim 2, wherein said stored instructions further comprise:

for receiving a data packet from said external networks, identifying one of said buffers reflecting ownership by one of said network applications, processing a data packet stored in said one of said buffers, and restoring said one of said buffers to a free status.

4. The multiprocessor computer system of claim 2, wherein said stored instructions further comprise:

for transmitting a data packet from said at least one network application, identifying a free one of said buffers, storing a data packet in said free buffer, and changing ownership status of said free buffer to reflect ownership by one of said plurality of network interfaces.

5. The multiprocessor computer system of claim 1, wherein said control processor receives raw load status data from said external networks and generates load distribution configuration data therefrom, said load distribution configuration data being accessible by said at least one switching processor, said at least one switching processor routing received ones of said data packets not directed to said at least one network application to a selected one of said external networks in accordance with information included in a header portion of said data packets and said load distribution configuration data.

6. The multiprocessor computer system of claim 1, wherein said at least one switching processor further provides periodic polling of corresponding ones of said network interfaces for detecting received ones of said data packets therein.

7. The multiprocessor computer system of claim 5, wherein said at least one switching processor further re-writes said routing information included in said header portion of said data packets to reflect said selected one of said external networks.

8. The multiprocessor computer system of claim 5, further comprising a shared memory including a connection table reflecting status of previously received ones of said data packets.

9. The multiprocessor computer system of claim 8, wherein said at least one switching processor accesses said connection table to determine correspondence between a received one of said data packets and said previously received ones of said data packets in determining said selected one of said external networks.

10. The multiprocessor computer system of claim 8, wherein said memory further comprises a configuration table containing said load distribution configuration data.

11. The multiprocessor computer system of claim 5, wherein said at least one switching processor further includes a switching program having an engine module containing stored instructions to be executed by said at least one switching processor, said stored instructions comprising the steps of:

polling a first one of said network interfaces for presence of a received data packet;

if a received data packet is present at said first one of said network interfaces, routing said received data packet to said selected one of said external networks; and

if a received one of said data packets is not present at said first one of said network interfaces, polling another one of said network interfaces for presence of a received data packet.

12. The multiprocessor computer system of claim 8, wherein said at least one switching processor further includes; a filter module having stored instructions to be executed by said at least one switching processor, said stored instructions comprising the steps of:

reading routing information from said header portion of said data packet;

accessing said load distribution configuration data stored in said shared memory;

selecting said selected one of said external networks based on said routing information and said load distribution configuration data;

modifying said data packet by re-writing said routing information to reflect said selected one of said external networks; and

sending said modified data packet to one of said plurality of network interfaces corresponding to said selected one of said external networks.

13. The multiprocessor computer system of claim 12, wherein said stored instructions of said filter module further comprises the steps of:

reading transport information from said header portion of said data packet; and

accessing connection status data stored in a connection table of said shared memory reflecting status of previously received -ones of said data packets, wherein, if said transport information indicates that said data packet corresponds to a previously received data packet, then said selecting step further comprises selecting said selected one of said external networks based on routing of said previously received data packet.

14. In a multiprocessor computer system comprising a plurality of symmetrical processors and a plurality of network interfaces each adapted to be coupled to respective external networks for receiving data packets from remote devices and sending data packets thereto via a particular communication protocol, a method for operating said computer system comprises the steps of:

configuring one of said plurality of processors as a control processor and others of said plurality of processors as switching processors, said control processor further having an operating system portion that includes a kernel memory and at least one network driver communicating with said plurality of network interfaces, at least one of said switching processors further including a network application executing thereon;

providing a buffer descriptor list accessible by said network application and said at least one network driver, said buffer descriptor list defining status of plural buffers provided in said kernel memory used for temporary storage of data packets transferred between said network application and said plurality of network interfaces via said network driver;

placing incoming data packets received by said network interfaces from said external networks directed to said at least one network application in selected ones of said buffers by said network driver for direct access by said network application; and

placing outgoing data packets transmitted from said at least one network application to said external networks in other selected ones of said buffers for direct access by said network driver.

15. The method of claim 14, further comprising the steps of:

disabling interrupts directed to said at least one switching processor; and

circling through said buffer descriptor list to monitor ownership status of said buffers.

16. The method of claim 14, wherein said step of placing incoming data packets further comprises:

identifying one of said buffers reflecting ownership by said network application, processing a data packet stored in said one of said buffers, and restoring said one of said buffers to a free status.

17. The method of claim 14, wherein said step of placing outgoing data packets further comprise:

identifying a free one of said buffers, storing a data packet in said free buffer, and changing ownership status of said free buffer to reflect ownership by one of said plurality of network interfaces.

18. The method of claim 14, further comprising the steps of:

providing load data to said control processor regarding load status of said external networks;

generating load distribution configuration data from said load data using said control processor and providing said load distribution configuration data for access by said data packet switching processors; and

routing a received data packets not directed to said network application using said switching processors to a selected one of said external networks in accordance with information included in a header portion of said data packet and said load distribution configuration data.

19. The method of claim 18, further comprising the step of re-writing said routing information included in said header portion of said data packets by said switching processors to reflect said selected one of said external networks.

20. The method of claim 18, further comprising the step of providing a connection table reflecting status of previously received ones of said data packets.

21. The method of claim 20, further comprising accessing said connection table by said switching processors to determine correspondence between said received one of said data packets and said previously received ones of said data packets in determining said selected one of said external networks.

22. The method of claim 18, further comprising providing a configuration table containing said load distribution configuration data.

23. The method of claim 14, wherein said particular communication protocol further comprises TCP/IP.

* * * * *

US006330610B1

(12) **United States Patent**
Docter et al.

(10) **Patent No.: US 6,330,610 B1**
(45) **Date of Patent: *Dec. 11, 2001**

(54) **MULTI-STAGE DATA FILTERING SYSTEM EMPLOYING MULTIPLE FILTERING CRITERIA**

(76) Inventors: **Eric E. Docter**, 1555 Wessex Ave., Los Altos, CA (US) 94024; **William D. Hofmann**, 1408 Carleton St., Berkeley, CA (US) 94702; **Paul E. Hurley**, 186 Forest Ave., #5A, Palo Alto, CA (US) 94301

(*) Notice: This patent issued on a continued prosecution application filed under 37 CFR 1.53(d), and is subject to the twenty year patent term provisions of 35 U.S.C. 154(a)(2).

Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **08/985,389**

(22) Filed: **Dec. 4, 1997**

(51) **Int. Cl.**^{7} **G06F 15/16**; G06F 15/177

(52) **U.S. Cl.** **709/229**; 709/202; 713/153; 713/154; 713/155; 713/201

(58) **Field of Search** 707/10, 2, 4; 364/137, 364/716.04, 724.011; 395/187.07, 200.59; 709/219, 249, 200, 229, 202; 713/200, 177, 201, 202, 154, 155, 150, 153, 161, 160, 162, 182; 705/55, 56

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,101,445	*	3/1992	Call et al.	382/263
5,187,787		2/1993	Skeen et al.	709/314
5,404,505	*	4/1995	Levinson	707/10

(List continued on next page.)

OTHER PUBLICATIONS

Rusty Russell, Linux 2.4 Packet Filtering HOWTO, May 2000.*

Logicon Message Dissemination System, LMDS 2.2, 1997.*

Protecting Your AS/400.*

Siyan Karanjit, Internet Firewalls and Network Secourity, Second Edition, 95.*

Pat Hensley, et al.; Proposal for an Open Profiling Standard; NOTE–OPS–FrameWork.html; Jun. 2, 1997; Version 1.

Joseph Reagle et al.;General Overview of the P3P Architecture; P3P Architecture Work Group; Oct. 22, 1997.

Pat Hensley, et al.; Implementation of OPS Over HTTP: NOTE–OPS–OverHTTP.html; Jun. 2, 1997; Version 1.

Pat Hensley, et al.; Standard Practices for OPS Systems; NOTE–OPS–Standard Practices.html; Jun. 2, 1997; Version 1.

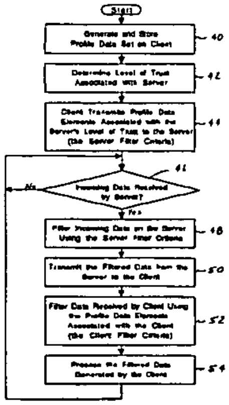
(List continued on next page.)

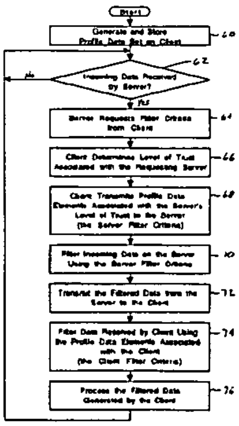
Primary Examiner—Mark H. Rinehart
Assistant Examiner—William C. Vaughn, Jr.
(74) *Attorney, Agent, or Firm*—Blakely, Sokoloff, Taylor & Zafman, LLP

(57) **ABSTRACT**

A data filtering system is provided that filters data in multiple stages. The system provides a first filter criteria to a first device. The first device uses the first filter criteria to generate a first set of filtered data. The system receives the first set of filtered data from the first device and filters the received data based on a second filter criteria, which is different from the first filter criteria. The filtering of the first set of filtered data generates a second set of filtered data. The first filter criteria and the second filter criteria can be included in a profile data set. The profile data set may be associated with a particular data recipient. The first filter criteria contains public profile data and the second filter criteria contains private profile data. The profile data set may contain data elements associated with a particular class of data recipients or a particular data recipient role. The data filtering system can be implemented such that the first device is an untrusted filtering device and the second device is a trusted filtering device.

36 Claims, 10 Drawing Sheets


Start
Generate and Store Profile Data Set on Client
40
Determine Level of Trust Associated with Server
42
Client Transmits Profile Data Elements Associated with the Server's Level of Trust to the Server (the Server Filter Criteria)
44
46
Incoming Data Received by Server?
No
Yes
Filter Incoming Data on the Server Using the Server Filter Criteria
48
Transmit the Filtered Data from the Server to the Client
50
Filter Data Received by Client Using the Profile Data Elements Associated with the Client (the Client Filter Criteria)
52
Process the Filtered Data Generated by the Client
54


Start
Generate and Store Profile Data Set on Client
60
62
Incoming Data Received by Server?
No
Yes
Server Requests Filter Criteria from Client
64
Client Determines Level of Trust Associated with the Requesting Server
66
Client Transmits Profile Data Elements Associated with the Server's Level of Trust to the Server (the Server Filter Criteria)
68
Filter Incoming Data on the Server Using the Server Filter Criteria
70
Transmit the Filtered Data from the Server to the Client
72
Filter Data Received by Client Using the Profile Data Elements Associated with the Client (the Client Filter Criteria)
74
Process the Filtered Data Generated by the Client
76

U.S. PATENT DOCUMENTS

5,493,658		2/1996	Chiang et al. .	
5,530,853		6/1996	Schell et al.	707/1
5,555,376	*	9/1996	Theimer et al.	709/229
5,557,798		9/1996	Skeen et al.	705/35
5,596,718	*	1/1997	Boebert et al.	395/187.01
5,606,668	*	2/1997	Shwed	395/187.01
5,608,662	*	3/1997	Large et al.	708/300
5,619,648		4/1997	Canale et al.	709/206
5,630,123	*	5/1997	Hogge	707/7
5,701,342	*	12/1997	Anderson et al.	380/4
5,740,423	*	4/1998	Logan et al.	707/10
5,781,550	*	7/1998	Templin et al.	370/401
5,787,253	*	7/1998	McCreery et al.	395/200.61
5,801,688	*	9/1998	Mead et al.	345/326
5,812,124	*	9/1998	Eick et al.	345/327
5,815,665	*	9/1998	Teper et al.	709/229
5,822,431	*	10/1998	Sprunk	713/150
5,826,268	*	10/1998	Schaefer et al.	707/9
5,842,040	*	11/1998	Hughes et al.	395/831
5,864,683	*	1/1999	Boebert et al.	709/249
5,867,651	*	2/1999	Dan et al.	709/203
5,867,799	*	2/1999	Lang et al.	707/1
5,872,847	*	2/1999	Boyle et al.	713/151
5,884,025	*	3/1999	Baehr et al.	713/201
5,884,033	*	3/1999	Duvall et al.	709/206
5,887,133	*	2/1999	Brown et al.	709/200
5,893,091	*	4/1999	Hunt et al.	707/3
5,898,830	*	4/1999	Wesinger, Jr. et al.	713/201
5,901,287	*	5/1999	Bull et al.	709/218
5,905,863	*	5/1999	Knowles et al.	709/206
5,910,987	*	6/1999	Ginter et al.	380/24
5,937,401	*	10/1999	Hillegas	707/2
5,951,638	*	9/1999	Hoss et al.	709/206
5,968,176	*	10/1999	Nessett et al.	713/201
5,978,475	*	11/1999	Schneier et al.	713/177
5,987,471	*	11/1999	Bodine et al.	707/103
6,005,565	*	12/1999	Legall et al.	345/327
6,009,422	*	12/1999	Ciccarelli	707/4
6,021,433	*	2/2000	Payne et al.	709/219
6,029,161	*	2/2000	Lang et al.	707/1
6,029,165	*	9/1998	Gable	707/3
6,067,569	*	5/2000	Khaki et al.	709/224
6,072,942	*	6/2000	Stockwell et al.	709/206
6,101,531		8/2000	Eggleston et al.	709/206
6,105,027	*	8/2000	Schneider et al.	707/9
6,105,132	*	8/2000	Fritch et al.	713/167

OTHER PUBLICATIONS

Staff Report: Public Workshop on Consumer Privacy on the Global Information Infrastructure; Dec. 1996.

I/Net Commerce Server 400 (Information); Protecting Your AS/400; 2000.

LOGICON; Logicon Message Dissemination System—LMDS 2.2 System Administrator's Guide: Arlington, VA.

PCT Written Opinion for Patent Application No. 98/25248 dated Sep. 13, 1999.

* cited by examiner

FIG. 1

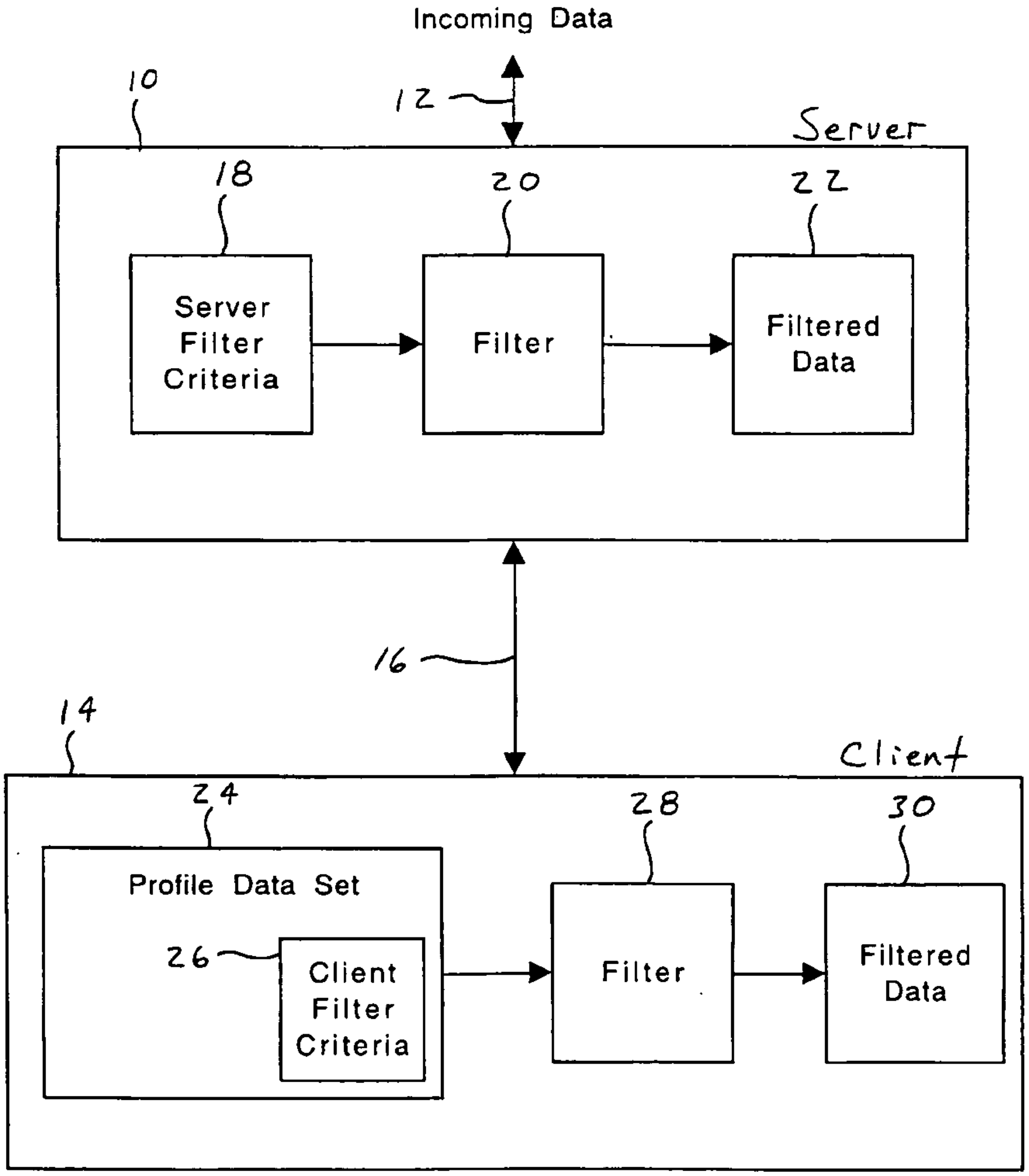

Incoming Data
10
12
Server
18
20
22
Server Filter Criteria
Filter
Filtered Data
16
14
Client
24
28
30
Profile Data Set
26
Client Filter Criteria
Filter
Filtered Data

FIG. 2

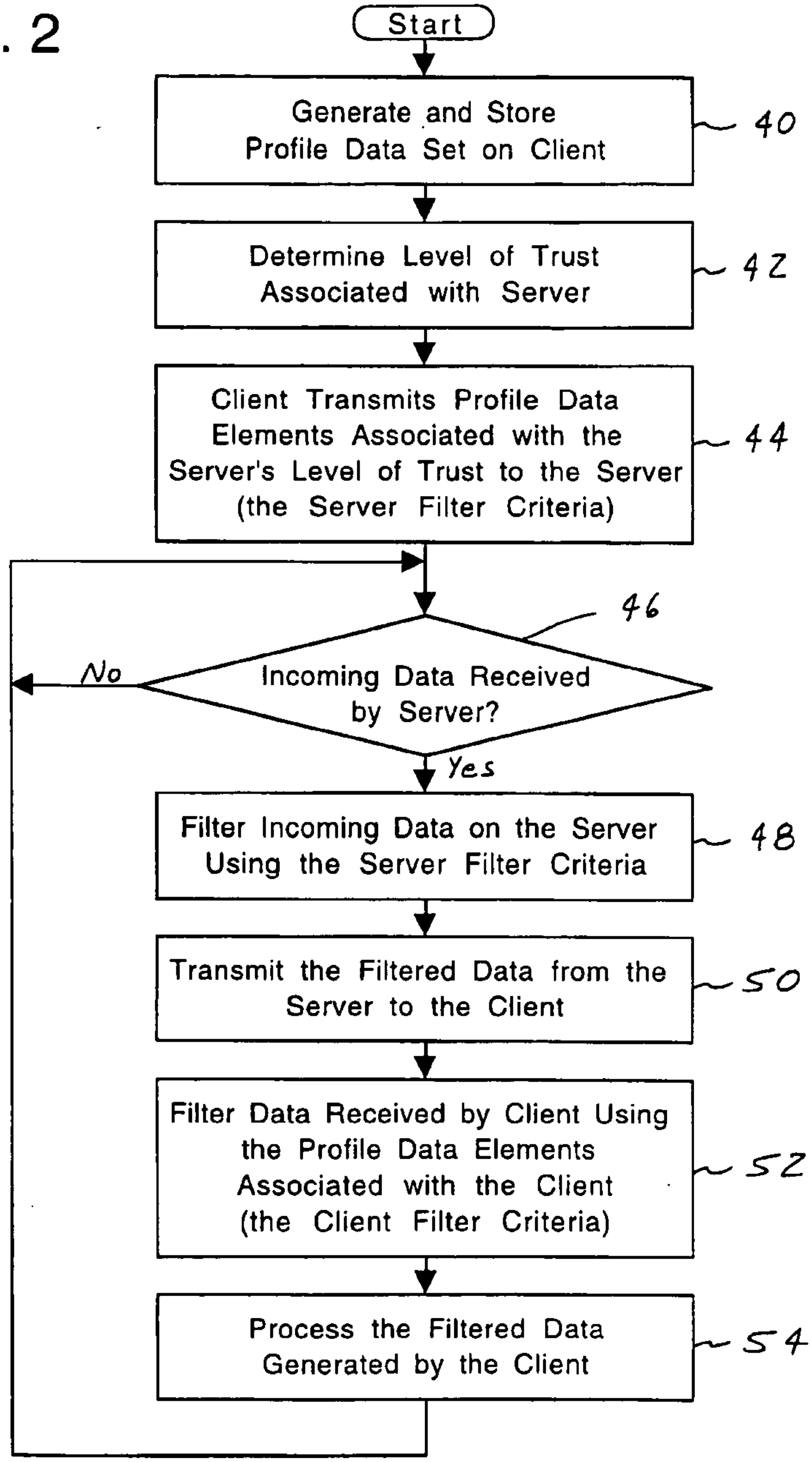

Start
Generate and Store Profile Data Set on Client
40
Determine Level of Trust Associated with Server
42
Client Transmits Profile Data Elements Associated with the Server's Level of Trust to the Server (the Server Filter Criteria)
44
46
Incoming Data Received by Server?
No
Yes
Filter Incoming Data on the Server Using the Server Filter Criteria
48
Transmit the Filtered Data from the Server to the Client
50
Filter Data Received by Client Using the Profile Data Elements Associated with the Client (the Client Filter Criteria)
52
Process the Filtered Data Generated by the Client
54

FIG. 3

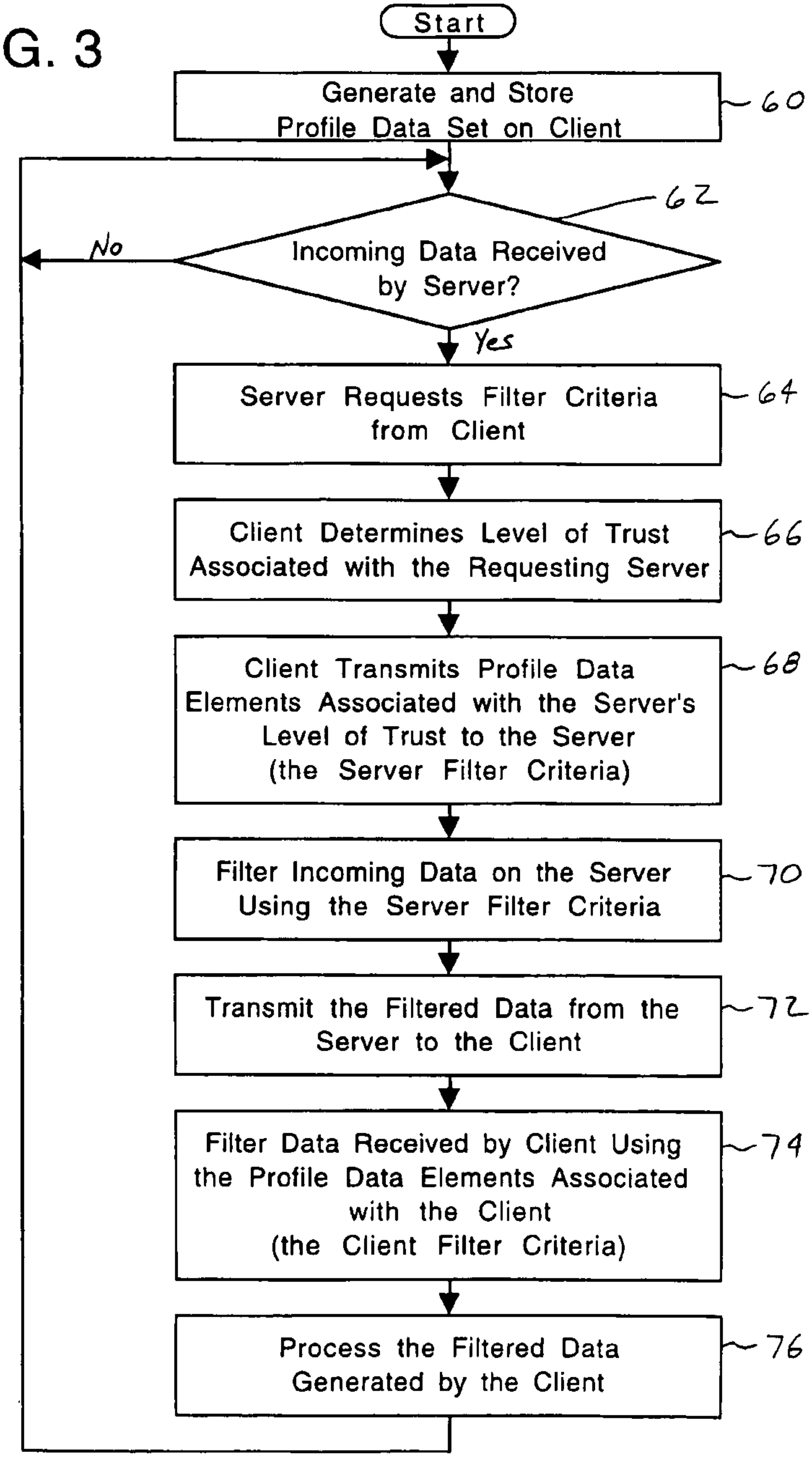

Start
Generate and Store Profile Data Set on Client
60
Incoming Data Received by Server?
62
No
Yes
Server Requests Filter Criteria from Client
64
Client Determines Level of Trust Associated with the Requesting Server
66
Client Transmits Profile Data Elements Associated with the Server's Level of Trust to the Server (the Server Filter Criteria)
68
Filter Incoming Data on the Server Using the Server Filter Criteria
70
Transmit the Filtered Data from the Server to the Client
72
Filter Data Received by Client Using the Profile Data Elements Associated with the Client (the Client Filter Criteria)
74
Process the Filtered Data Generated by the Client
76

FIG. 4

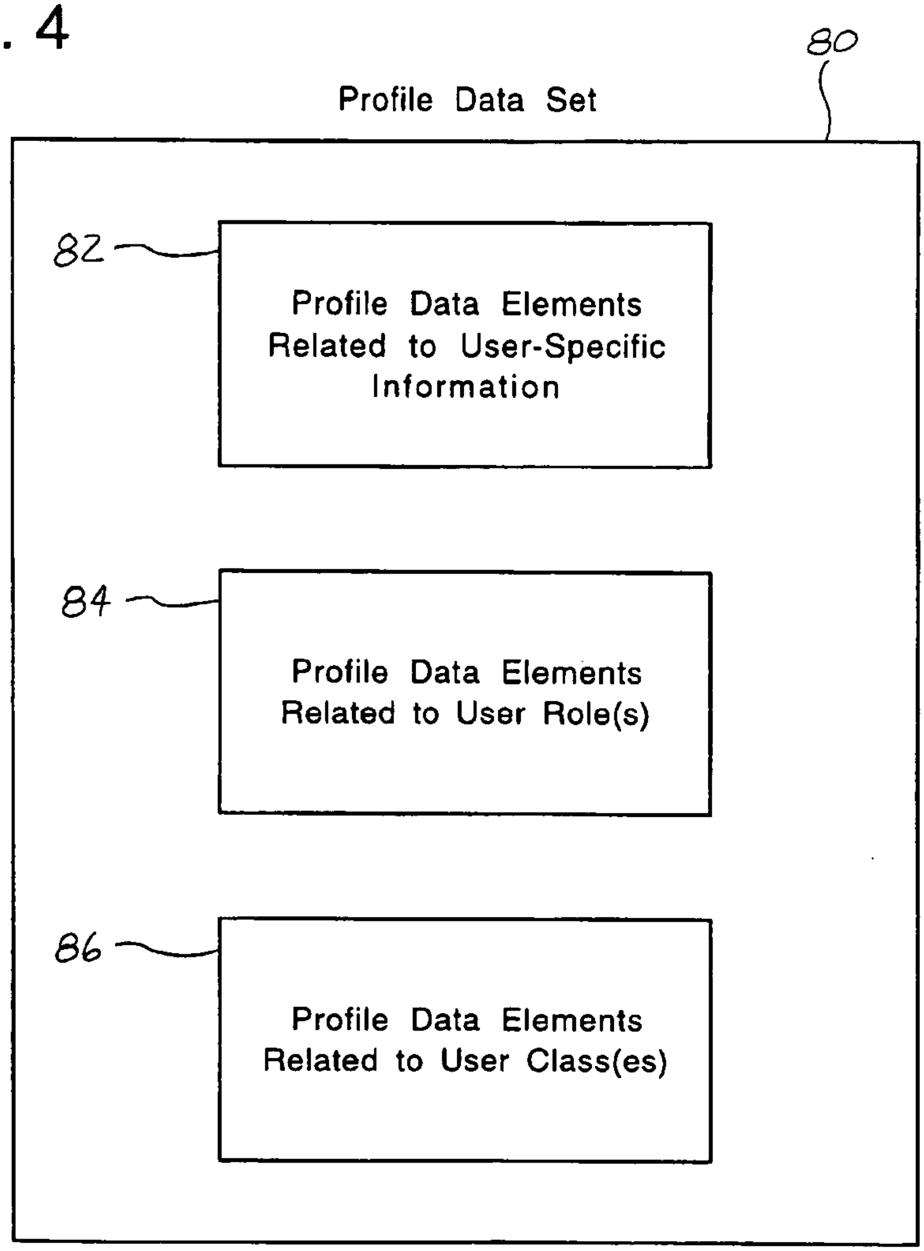

80
Profile Data Set
82
Profile Data Elements Related to User-Specific Information
84
Profile Data Elements Related to User Role(s)
86
Profile Data Elements Related to User Class(es)

FIG. 5

Attribute	Value	Privacy Characteristics
Name	John Doe	Public
Employer	Acme Corp.	Semi-Private
Occupation	Engineer	Semi-Private
Age	38	Semi-Private
Gender	Male	Public
Height	6-0	Semi-Private
Social Security No.	123-45-6789	Private
Marital Status	Married	Private

FIG. 6A

Attribute	Value
Name	John Doe
Gender	Male

FIG. 6B

Attribute	Value
Employer	Acme Corp.
Occupation	Engineer
Age	38
Height	6-0
Social Security No.	123-45-6789
Marital Status	Married

FIG. 7

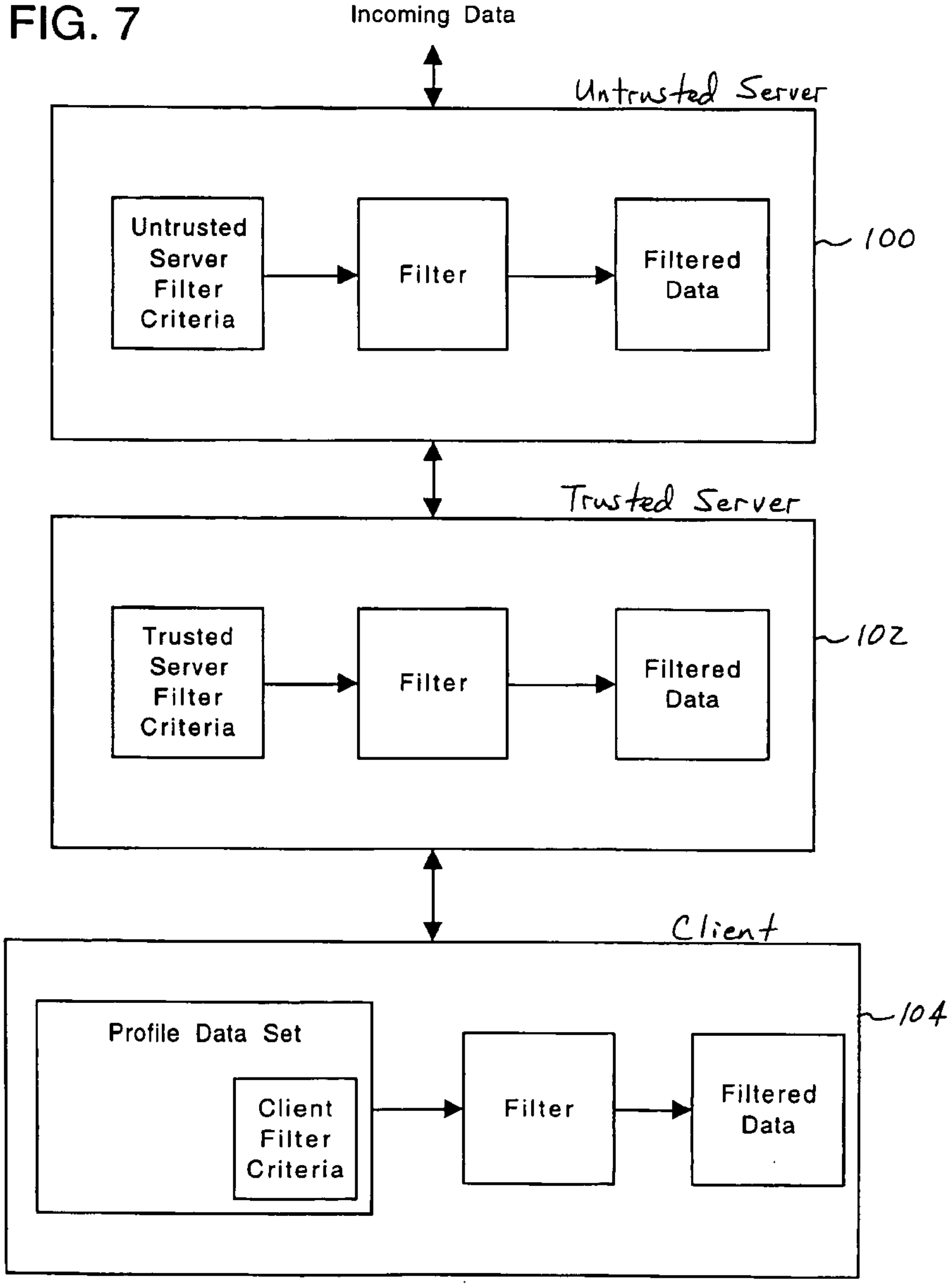

Incoming Data
Untrusted Server
Untrusted Server Filter Criteria
Filter
Filtered Data
100
Trusted Server
Trusted Server Filter Criteria
Filter
Filtered Data
102
Client
Profile Data Set
Client Filter Criteria
Filter
Filtered Data
104

FIG. 8A

Attribute	Value
Name	John Doe
Gender	Male

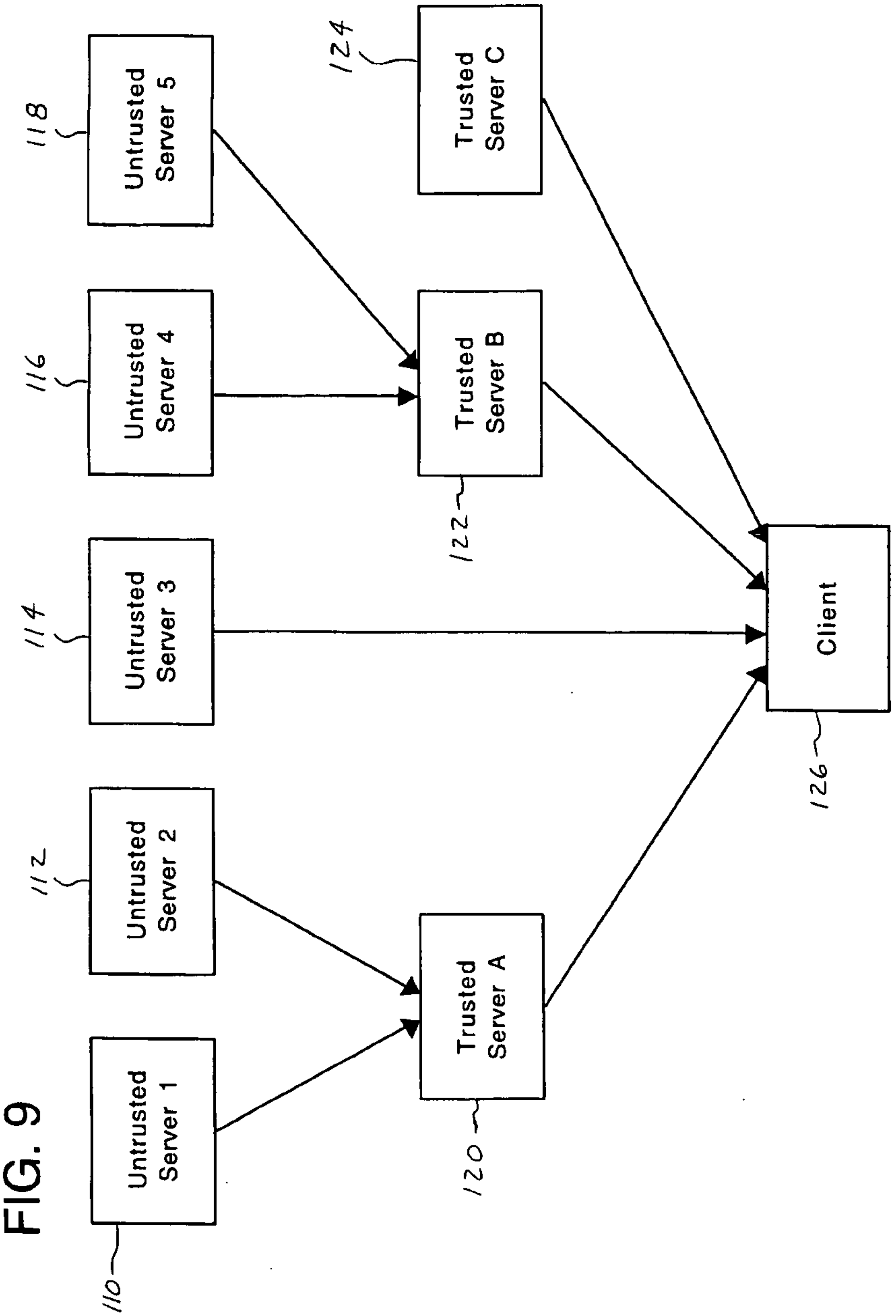
FIG. 8B

Attribute	Value
Employer	Acme Corp.
Occupation	Engineer
Age	38
Height	6-0

FIG. 8C

Attribute	Value
Social Security No.	123-45-6789
Marital Status	Married

FIG. 9


Untrusted Server 1
110
Untrusted Server 2
112
Untrusted Server 3
114
Untrusted Server 4
116
Untrusted Server 5
118
Trusted Server A
120
Trusted Server B
122
Trusted Server C
124
Client
126

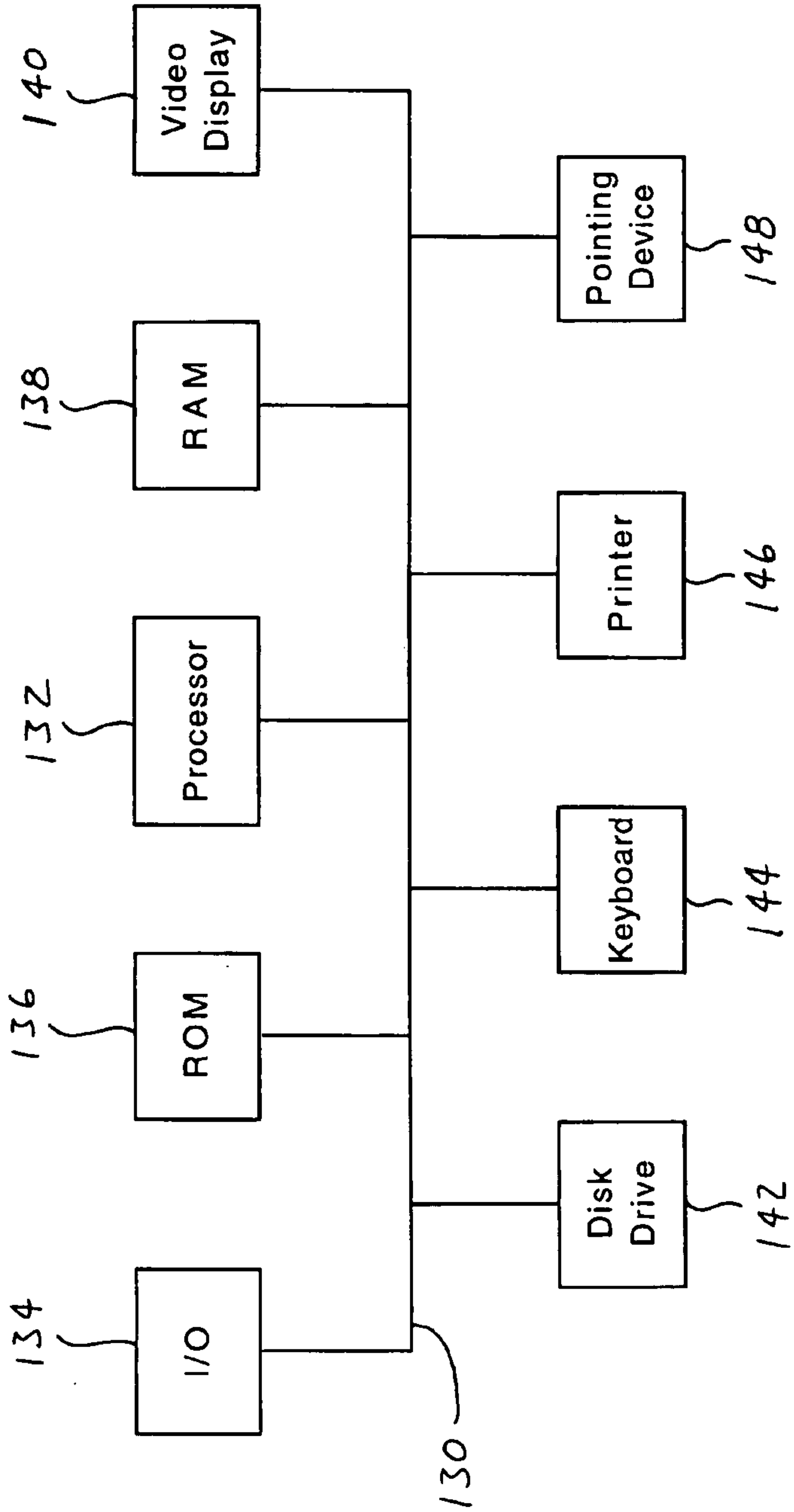
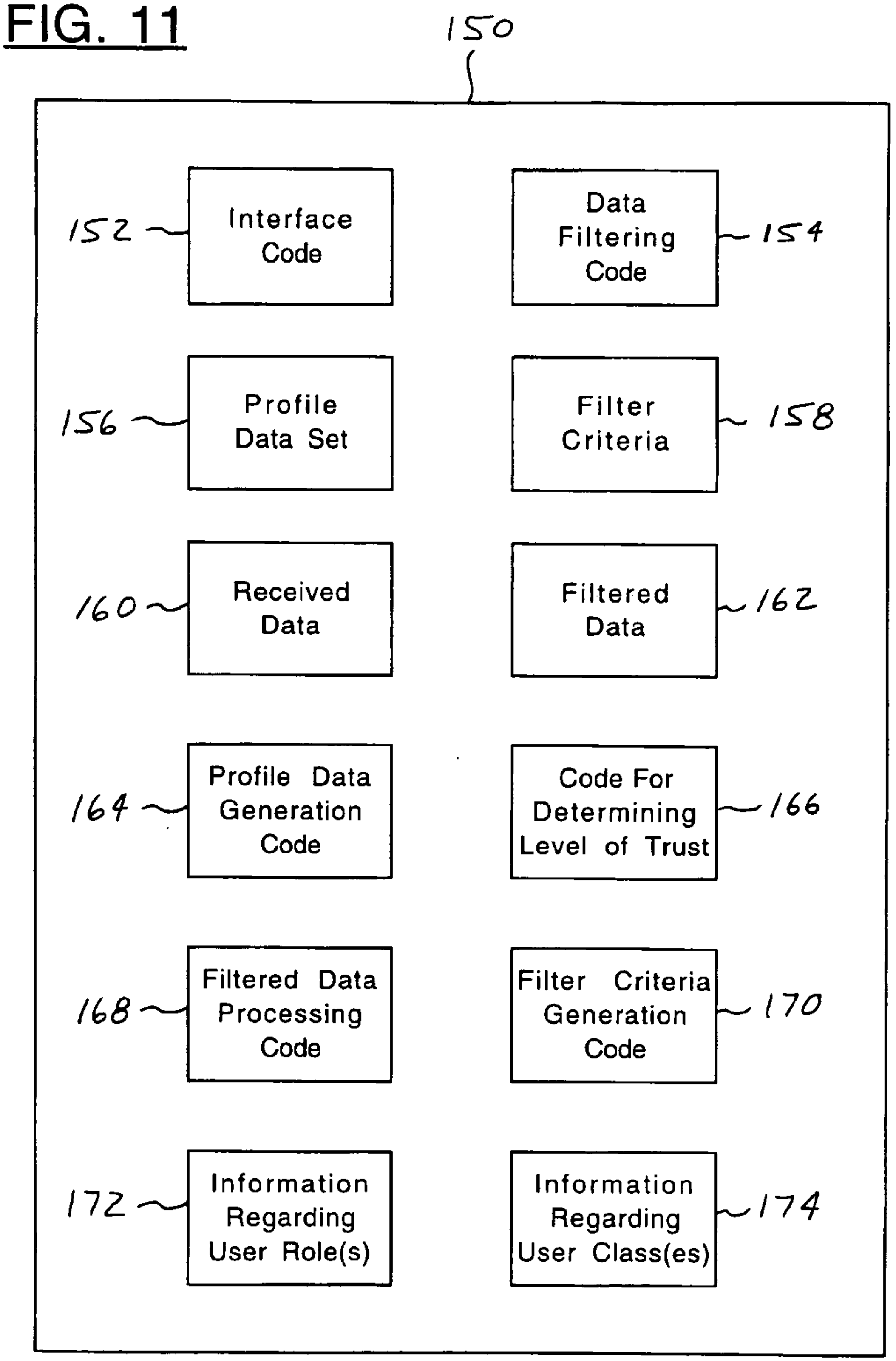

Video Display
140
RAM
138
Processor
132
ROM
136
I/O
134
Pointing Device
148
Printer
146
Keyboard
144
Disk Drive
142
130

FIG. 10

FIG. 11


150
152
Interface Code
154
Data Filtering Code
156
Profile Data Set
158
Filter Criteria
160
Received Data
162
Filtered Data
164
Profile Data Generation Code
166
Code For Determining Level of Trust
168
Filtered Data Processing Code
170
Filter Criteria Generation Code
172
Information Regarding User Role(s)
174
Information Regarding User Class(es)

MULTI-STAGE DATA FILTERING SYSTEM EMPLOYING MULTIPLE FILTERING CRITERIA

FIELD OF THE INVENTION

The present invention relates to a data filtering system. More specifically, the present invention provides a system capable of filtering data in multiple stages, with each stage of filtering using different filtering criteria.

BACKGROUND

The increased use of networks (such as the Internet) and networking technology has increased the quantity of data presented to individuals and organizations on a day-to-day basis. This data may be in the form of advertisements, news articles, and other information from any number of data sources. Although much of this data may be of interest to particular individuals and organizations, a significant portion of the data is generally of little or no value to the recipient. For example, the data may be related to a subject that is of no interest to the recipient or related to a type of product that the recipient does not use and does not intend to purchase.

Existing systems are available for selecting data to be provided to a particular user based on criteria that is supplied actively or passively by the user. These existing systems perform various filtering operations on a server to select the data to be provided to a particular user. Since these filtering operations are performed on a centralized server, the server must contain the necessary filtering criteria to select the data. These existing systems limit the effectiveness of the filtering operation because certain criteria necessary for proper filtering is confidential or private to the user and is not disclosed to the server. Since the server does not have this private information, it cannot adequately filter out all of the irrelevant data. For example, if a user does not indicate their age to the server, then the server cannot filter data that is directed at a particular age group. As a result, the user receives all data regardless of whether the data is relevant to a person in the user's age group.

Since the server is unable to filter data based on private criteria not provided to the server by the user, the user may receive a significant amount of irrelevant data. This irrelevant data is time consuming to review and creates a distraction from the user's normal work or activities. Since many servers that provide data filtering operations may not be trustworthy with respect to private information, many users are unwilling to provide private information to these servers. As a result, the user receives a significant amount of unwanted data.

Other known systems for filtering data perform all filtering operations on a client. These systems provide all data from all sources to the client, which then filters the data based on information provided by the user of the client. This approach significantly increases the amount of data received by the client and increases the bandwidth or transmission time required to transmit the data to the client from the data sources. The increase in data received by the client also increases the local storage requirements.

It is therefore desirable to provide a unified data filtering system capable of filtering out data that is not relevant to a particular user, without compromising the user's privacy.

SUMMARY OF THE INVENTION

The present invention is related to a system for filtering data in multiple stages without exposing private information to untrusted servers. In one embodiment of the invention, a first filter criteria is provided to a first device, which uses the first filter criteria to generate a first set of filtered data. The first set of filtered data is received from the first device and filtered based on a second filter criteria, which is different from the first filter criteria. The filtering of the data received from the first device generates a second set of filtered data.

In a particular embodiment of the invention, the first filter criteria and the second filter criteria are included in a profile data set.

In another embodiment of the invention, the first filter criteria contains public profile data and the second filter criteria contains private profile data.

Embodiments of the invention provide a profile data set that contains elements associated with a particular class of data recipients.

Other embodiments provide a profile data set that contains elements associated with a particular data recipient role.

In an embodiment of the invention, the first device is an untrusted filtering device and the second device is a trusted filtering device.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention is illustrated by way of example in the following drawings in which like references indicate similar elements. The following drawings disclose various embodiments of the present invention for purposes of illustration only and are not intended to limit the scope of the invention.

FIG. 1 illustrates an embodiment of a multi-stage data filtering system.

FIG. 2 is a flow diagram illustrating an embodiment of a procedure for performing multi-stage data filtering.

FIG. 3 is a flow diagram illustrating another embodiment of a procedure for performing multi-stage data filtering.

FIG. 4 illustrates an embodiment of a profile data set for use with the present invention.

FIG. 5 illustrates exemplary profile data elements related to user-specific information.

FIGS. 6A and 6B illustrate exemplary server filter criteria and client filter criteria generated from the profile data elements shown in FIG. 5.

FIG. 7 illustrates another embodiment of a multi-stage data filtering system.

FIGS. 8A–8C illustrate exemplary filter criteria for use in the multi-stage data filtering system shown in FIG. 7.

FIG. 9 illustrates another embodiment of a multi-stage data filtering system.

FIG. 10 illustrates an embodiment of a computer system that can be used with the present invention.

FIG. 11 illustrates an embodiment of a computer-readable medium containing various sets of instructions, code sequences, configuration information, and other data used by a computer or other processing device.

DETAILED DESCRIPTION

The following detailed description sets forth numerous specific details to provide a thorough understanding of the invention. However, those of ordinary skill in the art will appreciate that the invention may be practiced without these specific details. In other instances, well-known methods, procedures, protocols, components, and circuits have not been described in detail so as not to obscure the invention.

The present invention is related to a system capable of filtering data for a particular user (also referred to as a data recipient) without compromising that user's privacy. The invention provides a unified data filtering process such that data filtering is performed in multiple stages, with different filtering criteria used at each stage. In a first stage, data filtering can be performed by a server using non-private filtering criteria. The data that passes through the filter at the first stage continues to another data filter at a second stage. The second stage of filtering may be performed by a client or a more trusted server, thereby allowing filtering criteria containing private information about the user or organization. Any number of filtering stages may be utilized, depending on the number of servers or other devices located between the data source and the data recipient. By limiting private filtering criteria to trusted servers or clients, a significant amount of unwanted data is eliminated without compromising the user's privacy.

Throughout this detailed description of the invention, various embodiments are discussed that include a client coupled to one or more servers. The teachings of the present invention are applicable to any type of device containing a processor or a controller capable of executing instructions. Thus, the clients and servers discussed herein can be any type of computing device, including desktop or laptop computers, personal digital assistants (PDAs), set-top boxes, or devices containing embedded controllers or embedded processors. Further, any type of communication link and communication medium can be used to communicate information between two or more devices.

Particular data filtering procedures are described below that utilize a profile data set to generate filter criteria for servers and clients. However, it will be appreciated that any method or procedure for filtering data can be used with the present invention. Further, any number of filtering parameters or attributes may be used to filter data at any number of data filtering stages. Additionally, the present invention can be used with any type of data (e.g., text, graphics, product updates (such as software updates), or executable instructions) and with data received from any data source or sources.

FIG. 1 illustrates an embodiment of a multi-stage data filtering system. A server **10** receives incoming data on a communication link **12**. Communication link **12** may be a network communication link or any other link capable of communicating data between two or more devices. Server **10** communicates with a client **14** using a communication link **16**. Communication link **16** may be a link through a network or any other link capable of propagating data between server **10** and client **14**. Communication links **12** and **16** may use any type of communication medium, such as, but not limited to, wires, fiber optic cables, or wireless communication systems.

Server 10 includes server filter criteria **18**, which provides the filtering criteria used by a filter **20** to filter the incoming data. Server 10 may be an untrusted server with which users are unwilling to share private information. In this situation, server filter criteria **18** contains public information (i.e., public filtering criteria) that the user is willing to share with the server. Additional details regarding server filter criteria **18** and the operation of filter **20** are provided below. Filter **20** generates filtered data **22** as a result of applying server filter criteria **18** to the incoming data. Filtered data **22** is generally a subset of the incoming data received on communication link **12**. However, in certain situations, filtered data **22** is a null set of data if filter **20** removes (i.e., filters out) all of the incoming data. In other situations, all incoming data may pass through filter **20**, such that filtered data **22** contains all incoming data. Upon completion of the filtering operation performed by filter **20**, filtered data **22** is provided to client **14** using communication link **16**.

Client **14** contains a profile data set **24**, which includes client filter criteria **26**. In this embodiment, client **14** is trusted and, therefore, client filter criteria **26** may include private information that is not shared with server **10**. Profile data set **24** contains all profile data associated with a particular user or organization. This profile data is used to generate server filter criteria **18** and client filter criteria **26**. In the embodiment shown in FIG. 1, profile data set **24** contains server filter criteria **18** and client filter criteria **26**. In alternate embodiments, profile data set **24** may include filter criteria associated with a particular class of users or a particular role that a user performs. Additional details regarding profile data sets are provided below with respect to FIGS. 4–6.

Client **14** also includes a filter **28** that applies client filter criteria **26** to filtered data **22** received from server **10** on communication link **16**. Filter **28** generates a set of filtered data **30**, representing the incoming data that meets both server filter criteria **18** and client filter criteria **26**. Filtered data **30** is then provided to the user of client **14** for viewing or other processing. To maintain the privacy of the information contained in the profile data set, the results of the filtering process at any particular level of trust are not provided to a device or filtering process having a lower level of trust.

As shown in the filtering system of FIG. **1**, profile data set **24** is contained in client **14**. Thus, only the data that is public (i.e., not confidential or private) is shared with server **10**. The remaining filter criteria are stored on the client and is not exposed to or otherwise provided to the server. Thus, the single profile data set **24** provides a unified system for filtering incoming data on both server **10** and client **14**.

The embodiment of FIG. **1** represents a unified two-stage data filtering system. However, the teachings of the present invention may be applied to a data filtering system having any number of data filtering stages. An example of a unified three-stage data filtering system is illustrated in FIG. **7** and discussed below. Additionally, FIG. 1 shows a single client **14** coupled to server **10**. In other embodiments of the invention, a particular server may be coupled to multiple clients and contain separate filter criteria for each client that receives data from the server.

FIG. 2 is a flow diagram illustrating an embodiment of a procedure for performing multi-stage data filtering. The procedure illustrated in FIG. 2 may be used, for example, with the data filtering system illustrated in FIG. 1. At step **40**, a profile data set is generated and stored on a client (e.g., client **14** in FIG. 1). Additional details regarding the profile data set are discussed below with reference to FIGS. **4–6**. At step **42**, the procedure determines the level of trust associated with a server (e.g., server **10** in FIG. **1**). For example, a server located inside (i.e., on the corporate side) of a firewall may have a high level of trust and security, but a server located outside the firewall may be untrustworthy and is assigned a low level of trust. The level of trust associated with a particular server determines the type of profile data that is shared with that server for data filtering purposes. If a level of trust is not assigned to a particular server, then the server may be assigned a default level of trust (e.g., an untrusted server).

At step **44** of FIG. **2**, the client transmits profile data elements associated with the server's level of trust to the server. These profile data elements are referred to as the server filter criteria. The server filter criteria is stored within the server (e.g., in a register or other data storage mechanism). The server filter criteria may be stored temporarily or permanently. At step **46**, the procedure determines whether incoming data was received by the server. If no data was received, the procedure returns to step **46** to repeatedly test for incoming data. As an alternative to repeated testing for incoming data, the procedure may use a "trigger" that causes the procedure to continue to step **48** when incoming data is detected.

At step **48**, the procedure filters the incoming data on the server using the server filter criteria. Step **50** transmits the filtered data, if any, from the server to the client. At step **52**, the procedure filters data received by the client using the profile data elements associated with the client. These profile data elements are referred to as the client filter criteria. Finally, step **54** processes the filtered data, if any, generated by the client. This processing may include displaying the data to a user or notifying the user of the received data. If either the filtering performed by the server at step **48** or by the client at step **52** eliminates all data, then the procedure terminates without notifying the user.

FIG. **3** is a flow diagram illustrating another embodiment of a procedure for performing multi-stage data filtering. The procedure illustrated in FIG. **3** may be used, for example, with the data filtering system illustrated in FIG. **1**. The procedure of FIG. **3** is similar to the procedure discussed above with respect to FIG. **2**, but transmits profile data elements to the server after the receipt of incoming data instead of prior to the receipt of incoming data. At step **60**, a profile data set is generated and stored on the client. Step **62** determines whether incoming data has been received by the server. If incoming data has not been received, then the procedure returns to step **62** to continue testing for incoming data. Alternatively, a "trigger" can be used that causes the procedure to continue to step **64** when incoming data is detected.

When incoming data is received, the procedure continues to step **64**, in which the server requests filter criteria from the client. In response to the server's request for filter criteria, the client determines the level of trust associated with the requesting server at step **66**. At step **68**, the client transmits profile data elements associated with the server's level of trust to the server. These profile data elements are referred to as the server filter criteria. In a particular embodiment of the invention, the server discards the server filter criteria after filtering the received data. In an alternate embodiment of the invention, the server may store the server filter criteria for use with the next incoming data. In this alternate embodiment, the client may update the server with new server filter criteria each time the server filter criteria changes.

At step **70** of FIG. **3**, the incoming data is filtered on the server using the server filter criteria. Step **72** transmits the filtered data, if any, from the server to the client. At step **74**, the data received by the client is filtered using the profile data elements associated with the client (referred to as the client filter criteria). The filtered data, if any, generated by the client is then processed at step **76**. As discussed above, this processing may include displaying the filtered data to the user or notifying the user of the received data. If either the filtering performed by the server at step **70** or by the client at step **74** eliminates all data, then the procedure terminates without notifying the user.

Embodiments of the present invention execute the procedures described above with respect to FIGS. **2** and **3** continually (e.g., in a background mode). Therefore, the client and server(s) may exchange filter criteria, filtered data, and other information while the client is executing other applications or procedures.

FIG. **4** illustrates an embodiment of a profile data set **80** for use with the present invention. In one embodiment of the invention, a separate profile data set **80** is provided for each client (or each user). Profile data set **80** includes a set of profile data elements **82** that are related to user-specific information (e.g., age, occupation, or marital status). Profile data set **80** also includes a set of profile data elements **84** that are related to one or more user roles. A user role can be, for example, "professor" or "Vice President of Engineering." Profile data elements **84** related to a user role identify characteristics or attributes associated with that role, rather than an individual person. Therefore, all users performing a particular role may use profile data elements **84** rather than or in addition to entering those attributes along with their user-specific information. Furthermore, the attributes associated with a particular role can be updated once rather than updating each user's specific information. If a particular user performs multiple roles, then that user's profile data set **80** will contain profile data elements related to all of the roles performed by the user.

Profile data set **80** further includes a set of profile data elements **86** that are related to one or more user classes. A user class can be, for example, "marketing" or "engineers." Profile data elements **86** related to a user class identify characteristics or attributes associated with a class of users. Therefore, all users that are members of a particular class can use profile data elements **86** rather than entering those attributes along with their user-specific information. Additionally, the attributes associated with a particular class can be updated once rather than updating each member's specific information. If a particular user is a member of multiple classes, then that user's profile data set **80** will contain profile data elements related to all of the classes of which the user is a member. Additionally, a particular user may override the value associated with an attribute associated with a role or a class. For example, a role "Software Engineering Manager" may have an attribute "job level" with a value "grade 1." If a particular user performing the role of Software Engineering Manager has a job level "grade 2," that user's profile data set will contain an entry for the "job level"—"grade 2" pair that overrides the value provided by the role. Thus, the values associated with role or class attributes may operate as default values that can be changed by a user's profile data set.

As shown in FIG. **4**, profile data elements **84** related to user roles and profile data elements **86** related to user classes are stored within profile data set **80**. In alternative embodiments of the invention, a pointer or similar mechanism is provided in profile data set **80** that identifies a centralized storage location for the profile data elements related to user roles or user classes. The use of profile data elements related to user roles and user classes is optional. In alternative embodiments of the invention, profile data set **80** may include only profile data elements **82** related to user-specific information.

FIG. **5** illustrates exemplary profile data elements related to user-specific information (e.g., profile data elements **82** in FIG. **4**). The data elements shown in FIG. **5** are arranged into a table format for purposes of illustration. However, the actual data elements may be stored in any configuration using any data structure. The data elements in FIG. **5** include several attribute-value pairs (i.e., a value associated with each attribute). Additionally, a privacy characteristic is associated with each attribute-value pair. For example, the attribute "name" has a value "John Doe" and an associated privacy characteristic "Public." Thus, the user's name is John Doe and the user has made their name public. Public attributes are provided to all servers (whether the server is considered trustworthy or untrustworthy). The employer attribute has a value "Acme Corp." and has an associated privacy characteristic "Semi-Private." A "Semi-Private" privacy characteristic indicates that the attribute is only provided to trustworthy servers (i.e., not provided to untrustworthy servers). Trustworthy servers may be those servers located inside a corporate firewall and untrustworthy servers may be those servers located outside the corporate firewall. A third privacy characteristic, "Private," indicates that the attribute is only provided to clients, and is not provided to any server, whether trusted or untrusted. The example of FIG. 5 contains three different levels of privacy (Public, Semi-Private, and Private). However, in alternate embodiments of the invention, any number of privacy levels may be provided. As discussed in greater detail below, the number of privacy levels does not necessarily equal the number of filtering stages.

By using the profile data elements discussed above and assigning privacy characteristics to each attribute-value pair, the user is able to make an informed tradeoff between the privacy of the profile data and the bandwidth and local storage requirements. For example, if the user has a strong privacy interest, then only a few of the attribute-value pairs may be assigned a "Public" privacy characteristic. In this example, less profile data is exposed to untrusted servers, so additional data is received and processed by the client. In another situation, if the user desires a reduction in bandwidth and local storage requirements, many of the attribute-value pairs may be assigned a "Public" privacy characteristic. In this situation, more profile data is exposed to untrusted servers, but less data is received and stored by the client.

The privacy characteristics associated with a particular attribute-value pair can be determined by the user or the data provider. A default privacy characteristic may be provided for some or all of the attribute-value pairs. For example, a default privacy characteristic of "Private" may be associated with all attribute-value pairs to avoid exposing any private information about the user unless the user specifically changes the default setting.

Embodiments of the invention allow users to further limit the distribution of attribute-value pairs to particular types of servers. For example, a user of a particular brand of computer may only want the "Model Number" attribute to be provided to servers associated with the manufacturer of the computer. Thus, the "Model Number" may have a privacy characteristic of "Public", but the attribute-value pair is only distributed to servers associated with the particular manufacturer of the computer. The distribution of any attribute-value pair can be limited, regardless of the privacy characteristic. Additionally, a user may deactivate a particular attribute-value pair such that the attribute-value pair is not distributed to any server or client. The attribute-value pair remains deactivated until reactivated by the user. This deactivation provides a temporary way for a user to prevent filtering based on a particular attribute-value pair without permanently deleting the information from the profile data set.

FIGS. 6A and 6B illustrate exemplary server filter criteria and client filter criteria, respectively, generated from the profile data elements shown in FIG. 5. The server filter criteria shown in FIG. 6A contains two attribute-value pairs corresponding to the two "Public" entries shown in FIG. 5.

The server filter criteria shown in FIG. 6A does not include the privacy characteristics. The privacy characteristics are used to determine which servers or clients will receive a particular attribute-value pair. However, the privacy characteristics are not transmitted along with the filter criteria.

Using the exemplary filter criteria shown in FIG. 6A, a server is able to filter incoming data. For example, if the server receives incoming data (such as an advertisement or news article) targeted to male computer users over the age of 40, the server filter will allow the data to pass to the next data filtering stage because the server filter criteria for John Doe identifies that John Doe is male. Although the next data filtering stage will reject the data because John Doe is not over 40, the server is unaware of John Doe's age and cannot filter the data based on that attribute. Using the example filter criteria shown in FIG. 6A, the server is only capable of filtering incoming data based on the user's name and gender. If the user changes the privacy characteristic associated with attribute "Age" to "Public," then the server's filter criteria will include the attribute-value pair "Age—38". In this situation, the server will filter out the incoming data based on John Doe's age.

FIG. 6B contains six attribute-value pairs corresponding to the "Semi-Private" and "Private" entries shown in FIG. 5. In this example, two filtering stages are used, but three levels of privacy characteristics are provided. Therefore, two of the privacy characteristic levels are combined into a single filtering stage. For this example, "Public" entries are provided in the server filter criteria and "Semi-Private" and "Private" entries are provided in the client filter criteria. In an alternative embodiment, the "Public" and "Semi-Private" entries are provided in the server filter criteria and the "Private" entries are provided in the client filter criteria. Although FIG. 6B illustrates the client filter criteria separately from the profile data elements shown in FIG. 5, embodiments of the invention may read the client filter criteria directly from the profile data elements instead of generating a separate instance of the client filter criteria.

FIGS. 6A and 6B illustrate server filter criteria and client filter criteria having distinct attributes; i.e., no shared attributes. Thus, the server filter criteria and the client filter criteria are completely different from one another. However, in other embodiments of the invention, one or more of the attributes may be contained in two or more filter criteria. For example, the attribute "Age" may be contained in both the server filter criteria and the client filter criteria such that both the server and the client perform data filtering using the "Age" attribute. However, the server filter criteria and the client filter criteria do not generally share all attributes. Any two filter criteria are "different" if at least one data element is different between the two criteria (e.g., a different attribute or a different attribute value).

FIG. 7 illustrates another embodiment of a multi-stage data filtering system. The embodiment of FIG. 7 represents a unified three-stage data filtering system (untrusted server, trusted server, and client). As mentioned above, the teachings of the present invention may be applied to data filtering systems having any number of data filtering stages. The components contained within the servers and the client in FIG. 7 are similar to those discussed above with reference to FIG. 1. Untrusted server 100 receives incoming data from a data source (not shown) and filters the incoming data using an untrusted server filter criteria. The filtered data, if any, is then communicated from untrusted server 100 to trusted server 102. Trusted server 102 filters the received data using a trusted server filter criteria. The filtered data, if any, is then communicated from trusted server 102 to client 104. Client

104 filters the received data using a client filter criteria to generate a final set of filtered data. The filtering process may be terminated at any point if the output of a particular filter removes all data.

FIGS. 8A–8C illustrate exemplary filter criteria for use in the three-stage data filtering system shown in FIG. 7. FIGS. 8A–8C use the exemplary profile data elements shown in FIG. 5. FIG. 8A illustrates an untrusted server filter criteria (i.e., the attribute-value pairs having a privacy characteristic "Public"). FIG. 8B illustrates a trusted server filter criteria (i.e., the attribute-value pairs having a "Semi-Private" privacy characteristic). FIG. 8C illustrates a client filter criteria (i.e., the attribute-value pairs having a privacy characteristic "Private").

FIG. 9 illustrates another embodiment of a multi-stage data filtering system in which a client 126 receives data from multiple servers 110–124. A single profile data set is stored in client 126. Client 126 distributes various attribute-value pairs to the multiple servers based on the trustworthiness of the server and the privacy characteristics associated with each attribute-value pair. For example, untrusted servers 110 and 112 may receive an untrusted server filter criteria containing only "Public" attribute-value pairs, and trusted server 120 receives a trusted server filter criteria containing "Semi-Private" attribute-value pairs. Additionally, trusted server 124 may receive a trusted server filter criteria containing "Public" and "Semi-Private" attribute-value pairs. Untrusted server 112 may receive "Public" attribute-value pairs, while the "Semi-Private" and "Private" attribute pairs are filtered by client 126. Thus, client 126 may be filtering "Private" attribute-value pairs for some incoming data and filtering "Semi-Private" and "Private" attribute-value pairs for other incoming data.

It is not necessary that data filtering occur at every device through which the data passes. For example, untrusted servers 116 and 118 may receive "Public" attribute-value pairs, and the remaining "Semi-Private" and "Private" attribute-value pairs are filtered by client 126. In this example, the filtered data from untrusted servers 116 and 118 passes through trusted server 122 without any data filtering operation.

FIG. 10 illustrates an embodiment of a computer system that can be used with the present invention (e.g., as a client or a server). The various components shown in FIG. 10 are provided by way of example. Certain components of the computer in FIG. 10 can be deleted from the data filtering system for a particular implementation of the invention. The computer shown in FIG. 10 may be any type of computer including a general purpose computer.

FIG. 10 illustrates a system bus 130 to which various components are coupled. A processor 132 performs the processing tasks required by the computer. Processor 132 may be any type of processing device capable of implementing the steps necessary to perform the data filtering operations discussed above. An input/output (I/O) device 134 is coupled to bus 130 and provides a mechanism for communicating with other devices coupled to the computer. A read-only memory (ROM) 136 and a random access memory (RAM) 138 are coupled to bus 130 and provide a storage mechanism for various data and information used by the computer. Although ROM 136 and RAM 138 are shown coupled to bus 130, in alternate embodiments, ROM 136 and RAM 138 are coupled directly to processor 132 or coupled to a dedicated memory bus (not shown).

A video display 140 is coupled to bus 130 and displays various information and data to the user of the computer. A disk drive 142 is coupled to bus 130 and provides for the long-term mass storage of information. Disk drive 142 may be used to store various profile data sets and other data generated by and used by the data filtering system. A keyboard 144 and pointing device 148 are also coupled to bus 130 and provide mechanisms for entering information and commands to the computer. A printer 146 is coupled to bus 130 and is capable of creating a hard-copy of information generated by or used by the computer.

FIG. 11 illustrates an embodiment of a computer-readable medium 150 containing various sets of instructions, code sequences, configuration information, and other data used by a computer or other processing device. The embodiment illustrated in FIG. 11 is suitable for use with the data filtering system described above. The various information stored on medium 150 is used to perform various data filtering and data processing operations. Computer-readable medium 150 is also referred to as a processor-readable medium. Computer-readable medium 150 can be any type of magnetic, optical, or electrical storage medium including a diskette, magnetic tape, CD-ROM, memory device, or other storage medium.

Computer-readable medium 150 includes interface code 152 that controls the flow of information between various devices or components in a data filtering system. Interface code 152 may control the transfer of information within a device (e.g., between the processor and a memory device), or between an input/output port and a storage device. Additionally, interface code 152 may control the transfer of information from one device to another (e.g., the transfer of filtered data or profile data between a client and a server). Data filtering code 154 filters received data based on a particular filter criteria, as discussed above.

Computer-readable medium 150 also includes a profile data set 156 used to filter data and generate filter criteria. Profile data set 156 may include user-specific information, information related to user role(s), and/or information related to user class(es). Filter criteria 158 is used by the data filtering procedures described above. Received data 160 represents data that has been received by a particular device for filtering. Received data 160 may be filtered data from another device or may be unfiltered incoming data distributed by a third-party data source. Filtered data 162 represents the output of the data filtering process as applied to received data 160. If the filtering process filters out (i.e., removes) all received data 160, then filtered data 162 may be a null set.

Profile data generation code 164 typically resides on a client, and is used to generate profile data set 156. Profile data generation code 164 may be executed by a user of the client to generate or modify the various profile data attributes, values, and privacy characteristics contained in profile data set 156. Computer-readable medium 150 also includes code 166 for determining a level of trust associated with a particular device (such as a server). Typically, this code 166 is executed by a user of the client and may assign a default level of trust to a particular device if a level of trust is not otherwise assigned. For example, a default level of trust may be "untrusted," such that the device only receives profile data having a privacy characteristic of "Public."

Filtered data processing code 168 processes filtered data 162. For example, data processing code 168 may display filtered data 162 to a user, notify a user of the received data, or communicate filtered data 162 to the next device (e.g., transmit filtered data 162 from a server to a client). Filter criteria generation code 170 generates filter criteria based on information contained in profile data set **156** and the level of trust for a particular device as determined by code **166**. Typically, filter generation code **170** is executed by a client, which generates a filter criteria for a particular device. The filter criteria contains the attributes and values from profile data set **156** that correspond to the level of trust associated with the particular device. For example, an untrusted server may only receive attributes and values having a privacy characteristic of "Public." Therefore, the filter criteria for an untrusted server will not contain attributes and values having a privacy characteristic of "Semi-Private" or "Private."

Computer-readable medium **150** also includes information **172** regarding user role(s) and information **174** regarding user class(es). As discussed above, information relating to user roles and user classes identify characteristics or attributes associated with roles or classes, rather than an individual person. As shown in FIG. **11**, information **172** regarding user role(s) and information **174** regarding user class(es) may be stored separately from profile data set **156**. In alternate embodiments, information regarding user role(s) and class(es) may be stored within profile data set **156**.

FIG. **11** illustrates an exemplary computer-readable medium **150** containing various sets of instructions, code sequences, and other information that can be used by a data filtering system. However, in particular data filtering devices, one or more of the items illustrated in FIG. **11** may not be required. For example, in a computer-readable medium for use with an untrusted server that relies on a client for its filter criteria **158**, the computer-readable medium need not contain profile data set **156**, profile data generation code **164**, code **166** for determining level of trust, filter criteria generation code **170**, or information **172** and **174** regarding user role(s) and user class(es). In this example, the client maintains the profile data set, generates the filter criteria for the untrusted server, and communicates the filter criteria to the untrusted server. To maintain the privacy of the profile data set, the profile data set is typically stored only on the client.

Thus, a multi-stage data filtering system has been described that does not compromise a user's privacy. The system provides a filtering system that distributes multiple profile data elements to two or more data filtering stages, in which each data filtering stage may be performed by a different device or system.

From the above description and drawings, it will be understood by those of ordinary skill in the art that the particular embodiments shown and described are for purposes of illustration only and are not intended to limit the scope of the invention. Those of ordinary skill in the art will recognize that the invention may be embodied in other specific forms without departing from its spirit or essential characteristics. References to details of particular embodiments are not intended to limit the scope of the claims.

What is claimed is:

1. A method of filtering data, the method comprising:

providing from a trusted second device, a first filter criteria to an untrusted first device, the first filter criteria being non-private filter criteria and determined at least in part on trustworthiness of the untrusted first device, wherein the untrusted first device uses the first filter criteria to filter data received by the untrusted first device and generate a first set of filtered data;

the trusted second device receiving the first set of filtered data from the untrusted first device; and

filtering by the trusted second device the first set of filtered data based on a second filter criteria, the second filter criteria determined at least in part on trustworthiness of the trusted second device, wherein the filtering of the first set of filtered data generates a second set of filtered data, wherein the second filter criteria is different from the first filter criteria.

2. The method of claim **1** wherein the first filter criteria and the second filter criteria contain different filter characteristics.

3. The method of claim **1** wherein the first filter criteria and the second filter criteria are included in a profile data set.

4. The method of claim **3** wherein the first filter criteria contains public profile data.

5. The method of claim **3** wherein the second filter criteria contains private profile data.

6. The method of claim **3** wherein the profile data set contains at least one data element associated with user-specific information.

7. The method of claim **3** wherein the profile data set contains at least one data element associated with a user class.

8. The method of claim **3** wherein the profile data set contains at least one data element associated with a user role.

9. A method of filtering data, the method comprising the steps of:

filtering data in an untrusted first data filtering device based on a first filter criteria from a trusted second device, determined at least in part on trustworthiness of the first untrusted data filtering device so that private filter criteria is not provided to the untrusted first data filtering device, the filtering generating a first set of filtered data; and

providing at least a portion of the first set of filtered data to a trusted second data filtering device to filter the first set of filtered data based on a second filter criteria.

10. The method of claim **9** wherein the step of filtering at least a portion of the first set of filtered data in the trusted second data device generates a second set of filtered data.

11. The method of claim **9** wherein the first filter criteria and the second filter criteria contain different filter characteristics.

12. The method of claim **9** wherein the first filter criteria and the second filter criteria are included in a profile data set.

13. The method of claim **12**, in which the first data filtering device is untrusted such that the first filter criteria contains public profile data.

14. The method of claim **12** wherein the second filter criteria contains private profile data.

15. The method of claim **12** wherein the profile data set contains at least one data element associated with user-specific information.

16. The method of claim **12** wherein the profile data set contains at least one data element associated with a user role.

17. The method of claim **12** wherein the profile data set contains at least one data element associated with a user role.

18. A method of filtering data by an untrusted filtering device, the method comprising:

receiving a first filter criteria commensurate with the trustworthiness of the untrusted filtering device from a trusted data filtering device, the filtering criteria based at least in part according to trustworthiness of the untrusted filtering device and private filter criteria is not provided to the untrusted filtering device;

receiving incoming data from a data source;

filtering the incoming data using the first filter criteria to generate a first set of filtered data; and

providing the first set of filtered data to the trusted data filtering device, wherein the trusted data filtering device uses a second filter criteria based at least in part on trustworthiness of the trusted data filtering device to filter the first set of filtered data.

19. The method of claim 18 wherein the first filter criteria and the second filter criteria contain different filter characteristics.

20. The method of claim 18 wherein the first filter criteria and the second filter criteria are included in a profile data set.

21. The method of claim 20 wherein the second filter criteria contains private profile data.

22. The method of claim 21 wherein the method is performed by an untrusted filtering device.

23. A computer software product including a medium readable by a processor, the medium having stored thereon a sequence of instructions which, when executed by the processor of a trusted receiving device, cause the processor to:

provide to an untrusted first device from the trusted receiving device, a first filter criteria determined at least in part on trustworthiness of the untrusted first device, wherein the untrusted first device uses the first filter criteria to filter data received by the untrusted first device and generate a first set of filtered data;

the trusted receiving device receive the first set of filtered data from the untrusted first device; and

filter the first set of filtered data based on a second filter criteria to generate a second set of filtered data, the second filter criteria determined at least in part on trustworthiness of the trusted receiving device, wherein the second filter criteria is different from the first filter criteria, and wherein the filter criteria of the untrusted first device is non-private filter criteria.

24. The computer software product of claim 23 wherein the first filter criteria and the second filter criteria are included in a profile data set.

25. The computer software product of claim 24 wherein the first filter criteria contains public profile data.

26. The computer software product of claim 24 wherein the second filter criteria contains private profile data.

27. A computer software product including a medium readable by a processor of an untrusted receiving device, the medium having stored thereon a sequence of instructions which, when executed by the processor, cause the processor to:

receive a first filter criteria determined at least in part on trustworthiness of the untrusted receiving device from a trusted data filtering device wherein the first filter criteria is non-private filter criteria;

receive incoming data from a data source;

filter the incoming data using the received first filter criteria to generate a first set of filtered data; and

provide the first set of filtered data to the data filtering device, wherein the data filtering device uses a second filter criteria determined at least in part on trustworthiness of the trusted data filtering device to filter the first set of filtered data.

28. The computer software product of claim 27 wherein the first filter criteria and the second filter criteria contain different filter characteristics.

29. The computer software product of claim 27 wherein the first filter criteria and the second filter criteria are included in a profile data set.

30. The computer software product of claim 29 wherein the first filter criteria contains public profile data.

31. The computer software product of claim 29 wherein the second filter criteria contains private profile data.

32. A trusted data filtering apparatus comprising:

a data communication mechanism configured to provide a non-private filter criteria to an untrusted filtering device, wherein the untrusted filtering device uses the non-private filter criteria to generate a first set of filtered data, and wherein the data communication mechanism is further configured to receive the first set of filtered data from the untrusted filtering device; and

a data filter configured to filter the first set of filtered data based on a private filter criteria, wherein the filtering of the first set of filtered data generates a second set of filtered data,

wherein filtering criteria is configured at least in part according to trustworthiness of the untrusted filtering device so that private filter criteria is not provided to untrustworthy filtering devices.

33. The data filtering apparatus of claim 32 wherein the first filter criteria and the second filter criteria are included in a profile data set.

34. An untrusted data filtering apparatus comprising:

a data receiving mechanism configured to receive a first filter criteria from a trusted device and configured to receive incoming data from a data source;

a data filter configured to filter the incoming data using the first filter criteria and generate a first set of filtered data, the first filter criteria determined at least in part on the trustworthiness of the untrusted data filtering apparatus so that the untrusted data filtering apparatus does not receive private filter criteria; and

a data transmitting mechanism configured to transmit the first set of filtered data to the trusted device, wherein the trusted device uses a second filter criteria commensurate with a trustworthiness of the trusted device to filter the first set of filtered data.

35. The data filtering apparatus of claim 34 wherein the first filter criteria and the second filter criteria are included in a profile data set.

36. The method of claim 1 further including determining the trustworthiness of the trusted device.

* * * * *